INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Blockchain e Smart Contracts na Automatização das Interações de Parceiros de Negócio

FILIPE GABRIEL PINTO FERREIRA
Setembro de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Blockchain e *Smart Contracts* na Automatização das Interações de Parceiros de Negócio

Proof of Concept - Aluguer de Automóveis

Filipe Gabriel Pinto Ferreira

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Paulo Alexandre Gandra de Sousa

Porto, Setembro 2023

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 26 de setembro de 2023

Resumo

A tecnologia *blockchain* tem apresentado uma crescente tendência de aplicabilidade em diversos setores da indústria. O seu funcionamento assenta num sistema de informação descentralizado e prima pela considerável performance e elevada segurança.

Este projeto, realizado num contexto académico, pretende promover conhecimento acerca da tecnologia *blockchain*, e como esta juntamente com os *smart contracts* podem ser utilizados em sistemas industriais com múltiplos participantes, preservando as relações de mutualismo.

Para tal, elaborou-se uma investigação cuidada dos conceitos da tecnologia, de casos industriais reais da sua utilização, e uma análise de projetos *blockchain* possíveis de utilizar. O processo de negócio concebido é relativo ao aluguer de automóveis, e de que forma, com recurso à *blockchain* várias empresas do mesmo ramo industrial conseguem partilhar informação relativa aos seus ativos.

A solução final desenvolvida resulta numa prova de conceito totalmente funcional utilizando o projeto Hyperledger Fabric, com um conjunto de *smart contracts* que atendem ao processo de negócio.

Os resultados obtidos são considerados positivos, dada a elevada componente de investigação inerente à temática e pormenorização da implementação da solução desenvolvida, que representa um sólido suporte para soluções futuras. Cumpriu-se com os objetivos propostos.

Palavras-chave: *Blockchain*, *Smart Contracts*, Mutualismo, Prova de Conceito, Hyperledger Fabric.

Abstract

Blockchain technology has shown a growing trend of applicability in various industry sectors. Its operation is based on a decentralized information system and excels in considerable performance and high security.

This project, carried out in an academic context, aims to promote knowledge about the blockchain technology, and how it, together with smart contracts, can be used in industrial systems with multiple participants, preserving mutualistic relationships.

For this purpose, a careful investigation of the technology's concepts, real industrial cases of its use, and an analysis of blockchain projects that could be used were carried out. The designed business process is related to car rental, and how, using blockchain, several companies in the same industrial sector can share information regarding their assets.

The final solution developed results in a fully functional proof of concept using the Hyperledger Fabric project, with a set of smart contracts that serve the business process.

The results obtained are considered positive, given the high level of research inherent to the theme and the detailed implementation of the developed solution, which represents a solid support for future solutions. The proposed objectives were accomplished.

Keywords: Blockchain, Smart Contracts, Mutualism, Proof of Concept, Hyperledger Fabric.

Agradecimentos

Agradeço aos meus **pais** e **irmão**,
Pelo suporte e apoio incondicional proporcionado ao longo destes difíceis anos enquanto trabalhador-estudante.

Agradeço ao **ISEP** e seus **docentes**,
Que ao longo destes anos de licenciatura e mestrado, surgiram no meu percurso académico e contribuíram na sua progressão.

Agradeço em especial ao professor **Paulo Gandra**,
Por aceitar ser meu orientador, pelo encorajamento e suporte no desenvolvimento de um trabalho numa área desconhecida para mim.

Agradeço a **todos**, que direta ou indiretamente contribuíram para a realização deste trabalho e sucesso académico.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Acrónimos e Símbolos

Lista de Acrónimos

ACL	*Access Control List* (Lista de Controlos de Acesso)
BC	*Blockchain* (Cadeia de Blocos)
CA	*Certificate Authority* (Autoridade Certificadora)
CLI	*Command-line Interface*
DAO	*Descentralized Autonomous Organizations* (Organizações Autónomas Descentralizadas)
DApp	*Descentralized Application* (Aplicação Descentralizada)
DeFi	*Descentralized Fincance* (Finanças Descentralizadas)
DID	*Descentralized Identifier* (Identificador Descentralizado)
DLT	*Descentralized Ledger Technology* (Tecnologia de *Ledger* Distribuído)
ETC	Ether (Criptomoeda oficial do Ethereum)
FFE	*Fuzzy Front End*
GO	Linguagem de programação Golang
HLF	Hyperledger Fabric
IoT	*Internet of Things* (*Internet* das coisas)
ISEP	Instituto Superior de Engenharia do Porto
JSON	*JavaScript Object Notation*
KYC	*Know Your Customer* (Conhecer o Cliente)
LDAP	*Lightweight Directory Access Protocol*
MEI	Mestrado Engenharia Informática
MSP	*Member Service Provider*
NFT	*Non-fungible tokens* (*Tokens* não fungíveis)
NPD	*New Product Development* (Desenvolvimento de Novo Produto)

ORG	*Organization* participante na *blockchain* HLF
P2P	*Peer-to-Peer network* (rede Ponto a Ponto)
PBFT	*Practical Byzantine Fault Tolerance* (Prática de Tolerância a Falhas *Byzantine*)
PKI	*Public Key Infrastructure* (Infraestrutura de Chaves Públicas)
POC	*Proof of Concept* (Prova de Conceito)
POS	*Proof-of-Stake* (Prova de Participação)
POW	*Proof-of-Work* (Prova de Trabalho)
QR	Quick Response Code (Código de Resposta Rápida)
SC	*Smart Contracts* (Contratos Inteligentes)
SHA256	*Secure Hash Algorithm* 256 *bits* (Algoritmo de *Hash* Seguro de 256 *bits*)
SPOF	*Single Point of Failure* (Ponto Único de Falha)
TMDEI	Unidade Curricular Tese/Dissertação/Estágio
TPS	Transações por Segundo
USB	*Universal Serial Bus*
YAML	*Yet Another Markup Language*

Lista de Símbolos

$	*United States Dollar*

1 Introdução

O presente documento descreve a elaboração de um *proof of concept* (POC) para um cenário industrial, utilizando a tecnologia *Blockchain*.

Este capítulo contextualiza a realização deste POC e apresenta sucintamente os conceitos de *Blockchain* e *Smart Contracts*. Pretende descrever a motivação que levou ao desenvolvimento desta proposta de solução e os seus objetivos. Por fim, é apresentada a estrutura do documento.

1.1 Enquadramento/Contexto

A unidade curricular de TMDEI, inserida no Mestrado em Engenharia Informática (MEI) do ramo Engenharia de Software do ISEP, tem como objetivo propor aos alunos a elaboração de um trabalho de pesquisa e desenvolvimento conducente à preparação de uma dissertação técnico-científica, ou a realização de um estágio num ambiente empresarial, com o objetivo de aplicar, por parte dos estudantes, todos os conhecimentos e competências curriculares adquiridas.

Habitualmente, o conceito *Blockchain* surge associado às moedas digitais (*cryptocurrencies*), sendo a *Bitcoin* o projeto *open-source* criador da tecnologia, que surgiu em 2009 pelo *developer* com o pseudônimo Satocshi Nakamoto (Bitcoin, 2022). Esta tecnologia consiste num tipo de *Descentralized Ledger Technology* (DLT) (R3, 2022), com o registo descentralizado e imutável das transações efetuadas por todos os participantes, sem que estes se conheçam (Crosby, et al., 2016).

Para além do mercado das *cryptocurrencies*, a tecnologia apresenta uma ampla aplicabilidade na indústria, com uma evolução notória e previsão de crescimento bastante promissora: "*The global blockchain technology market is expected to witness a compound annual growth rate of 85.9% from 2022 to 2030 to reach USD 1,431.54 billion by 2030*" (Grand View Research, 2022).

A tecnologia *blockchain* apresenta diversos benefícios essenciais que justificam a emergente aposta, todas as transações são imutáveis, instantâneas e transparentes, com recurso a criptografia para promover a segurança. O sistema é descentralizado e a autenticidade das transações é garantida pelos participantes (Afreen, 2022).

Atualmente, esta tecnologia é aplicada em serviços governamentais, serviços de saúde, retalho, cadeias transportes e logísticas, serviços financeiros, entre outros. A Figura 1 apresenta uma distribuição de casos de uso de DLT identificados.

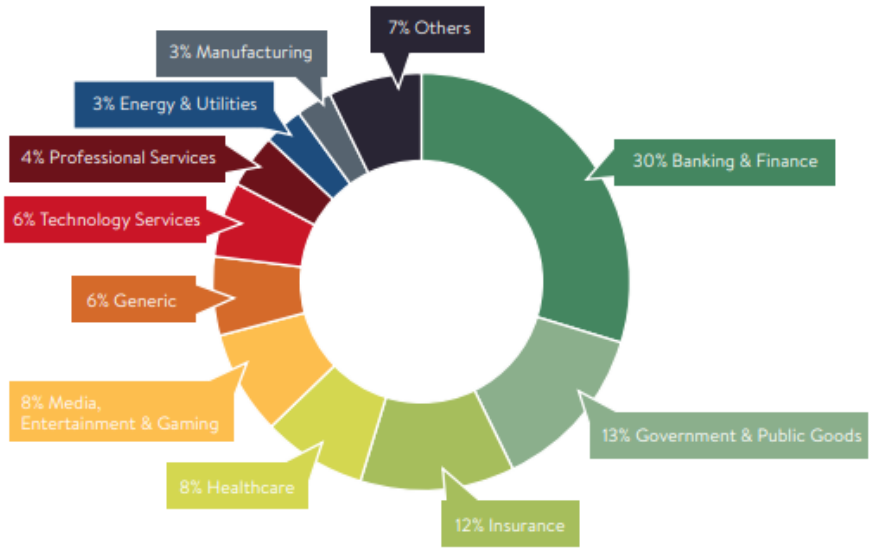


Figura 1 – Diagrama circular de casos de uso de DLT (Hileman & Rauchs, 2017).

Na figura acima (Figura 1), a indústria de *Banking* e *Finance* destaca-se como a que regista o maior número de casos de uso de DLT, seguido dos serviços públicos, governamentais e, mercado dos seguros.

Os *Smart Contracts* são contratos programáveis que são executados automaticamente numa determinada *blockchain* com o objetivo de garantir determinadas condições, ou seja, termos de um acordo entre participantes na *chain* são transformados em código computacional (Ethereum, 2022).

O principal foco desta dissertação é elaborar um POC que permita automatizar as interações entre os parceiros intervenientes de um determinado cenário de negócio. Para tal, elegeu-se a indústria do aluguer de automóveis.

Assim, com a elaboração deste documento pretende-se contribuir para o desenvolvimento de conhecimento na área de *blockchain*, perspetivar a sua aplicação no quotidiano e explorar algumas das vantagens principais da tecnologia, sendo a principal motivação para o projeto a curiosidade e interesse pessoal pela tecnologia.

1.2 Descrição do Problema

Pretende-se desenvolver um POC com o propósito otimizar as interações entre parceiros de um determinado negócio, onde a troca de informações entre várias entidades seja vital para o sucesso das operações. Para tal, definiu-se um caso de estudo baseado num contexto de negócio real, o aluguer de automóveis.

O aluguer de automóveis apresenta diversos cenários de interação cliente-empresa, por exemplo:

- Reserva do automóvel por parte do cliente;
- Levantamento do automóvel pelo cliente, na empresa de aluguer;
- Utilização do automóvel pelo cliente, nas condições contratadas.

Geralmente, nestas etapas do negócio estão intrínsecas diversas condições de adesão a cumprir pelos clientes, bem como transações entre a empresa prestadora do serviço e outras entidades externas. Contudo, o cenário a explorar neste trabalho pretende expandir atualidade desta indústria, isto é, para além de incluir os consumidores (clientes) e empresa prestadora do serviço, pretende-se simular a integração de diversas empresas de aluguer automóvel numa única *blockchain*, permitindo a partilha de ativos, preservando relações de mutualismo entre todos.

A Figura 2 esquematiza uma possível interação entre parceiros numa rede *blockchain* com a utilização de *smart contracts*.

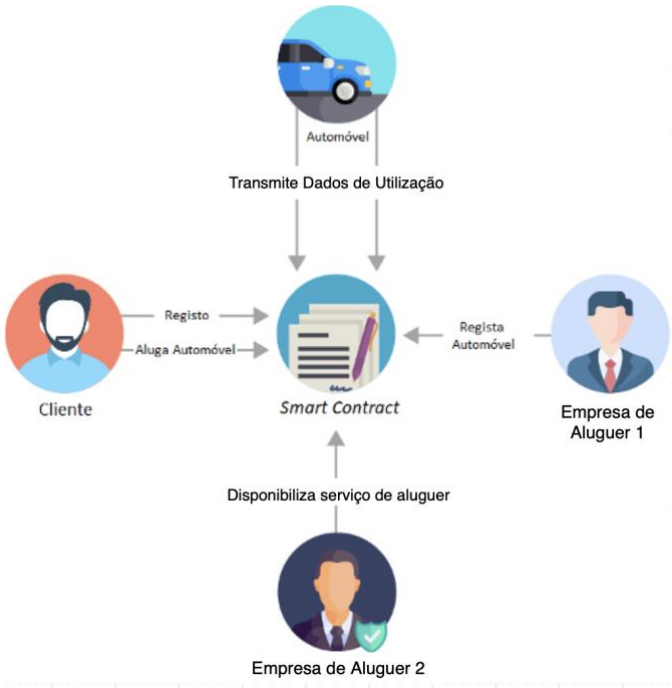


Figura 2 – Interação entre diversos participantes na *blockchain* com *smart contracts* (Serhii & Kotik, 2021).

A *blockchain* a desenvolver deve apresentar as seguintes características:

- Tipo privada, os participantes devem de ser identificáveis;
- Gestão de permissões;
- Elevado nível de performance com grande volume de informação;
- Garantir a confidencialidade dos dados.

Os *smart contracts* são utilizados na *blockchain* como validadores das complexas regras do negócio, que são executados a cada transação com o objetivo de a validarem. Por exemplo, o cliente pretende alugar um determinado automóvel junto da uma empresa, esta recolhe os dados necessários e estabelece um contrato de utilização com o cliente, os *smart contracts* serão constituídos por lógica capaz de validar os dados do cliente, bem como assegurar que as condições de utilização dos veículos foram devidamente cumpridas.

A utilização da tecnologia *blockchain* não é estritamente necessária no caso de estudo definido, no entanto pretende-se estudar uma alternativa aos sistemas tradicionais atualmente em operação. Esta poderá revelar-se vantajosa na otimização das interações dos participantes, promover a segurança e integridade das transações inerentes, recorrendo a mecanismos de criptografia, e os *smart contracts* apresentam uma aplicabilidade vasta na garantia de regras de negócio pré-definidas, de forma a preservar as relações de mutualismo entre os participantes.

Assim, este é o problema que o trabalho de engenharia descrito nas seguintes secções visa solucionar.

1.3 Objetivos

Com vista no problema já apresentado (secção 1.1), a realização deste trabalho tem como principais objetivos:

1. Adquirir e desenvolver conhecimentos na tecnologia *blockchain*, assim como, no desenvolvimento de soluções utilizando projetos *blockchain* disponíveis;
2. Definir, desenvolver e avaliar um POC relativo a um processo de negócio da indústria atual, tendo por base a tecnologia *blockchain* e a utilização de *smart contracts*:
3. Identificar boas práticas para a utilização de *blockchain* e *smart contracts* em redes com múltiplos participantes.

Relativamente ao primeiro objetivo, sendo a tecnologia *blockchain* emergente no mercado, há necessidade de obter conhecimento dos seus conceitos, de modo, a explorar-se um projeto *blockchain* adequado às necessidades do problema.

De seguida, pretende-se elaborar um POC com um modelo de domínio idealizado para a indústria do aluguer automóvel, a utilizar na base do desenvolvimento da rede *blockchain*. Esta deve possuir diversos participantes, como, clientes, várias empresas de aluguer automóvel, automóveis (possíveis elementos IoT), entre outros, com permissões distintas. Todas as transações devem ser registadas e validades com auxílio de um conjunto de *smart contracts.*

Contudo, é determinante atender às boas praticas de *design* e desenvolvimento de *software* no contexto de *blockchain* e *smart contracts*, de forma a conceber uma solução sólida e futuramente providenciar conhecimento.

Como objetivo opcional, poderá ser estudada a integração de dispositivos IoT (*Internet of Things*), como sensores para monitorização da utilização dos veículos pelos clientes.

Note-se que, alguns dos conceitos usados nesta secção são devidamente definidos nos próximos capítulos (capítulo 2).

1.4 Estrutura do Documento

Este documento está estruturado com os seguintes capítulos:

- **Introdução:** apresenta sucintamente o problema e objetivos do trabalho;
- **Estado da Arte:** explica detalhadamente os conceitos das áreas exploradas neste projeto, tecnologias existentes e casos reais da aplicação da tecnologia na indústria;
- **Análise de Valor:** descreve o valor da solução proposta, justifica a escolha do tipo de sistema a desenvolver e a sua tecnologia base;
- **Hyperledger Fabric:** descreve os principais conceitos da tecnologia Hyperledger Fabric;
- **Análise e Desenho da Solução:** detalha as especificidades do POC, o processo de negócio e os aspetos de *design*;
- **Implementação da Solução:** apresenta todo o processo de desenvolvimento da solução, tanto a configuração da rede como implementação dos *smart contracts*;
- **Experimentação e Avaliação:** testes e análise do sistema desenvolvido;
- **Conclusão:** encerra o documento com considerações acerca do trabalho desenvolvido;
- **Anexos:** conteúdo de suporte aos capítulos do documento**.**

2 Estado da Arte

Este capítulo pretende descrever o estado da arte relativo ao projeto desenvolvido, explicando-se o contexto tecnológico em que se enquadra, as soluções tecnológicas existentes no mercado, a área de negócio em que se insere, bem como outras áreas de negócio onde a tecnologia é aplicada. Assim, conceitos como *Blockchain* e *Smart Contracts* serão explicados com especial detalhe, bem como projetos tecnológicos considerados.

2.1 Fundamentos do *Blockchain*

A tecnologia *Blockchain* (BC) consiste num registo de transações de forma descentralizada, através de uma rede *Peer-to-Peer* (P2P). Todos os registos na rede BC ocorrem de forma eficiente e com um custo reduzido. A posse da informação das transações é partilhada pelos participantes, tornando-as imutáveis, ou seja, não podem ser alteradas ou eliminadas (Shrivas & Yeboah, 2017).

A Figura 3 esquematiza um sistema de informação centralizado e outro descentralizado.

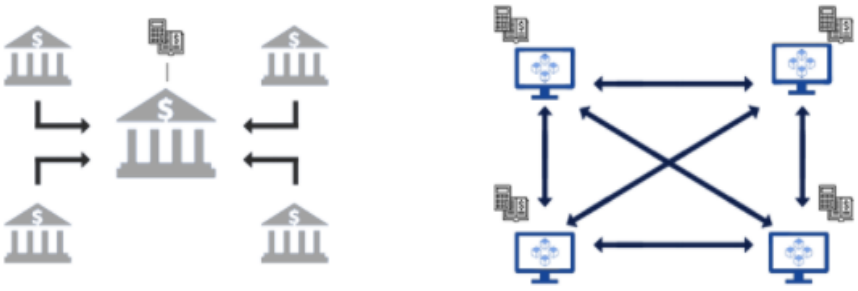


Figura 3 – Centralização e descentralização (Stably, 2019).

À esquerda da figura (Figura 3), a informação é centralizada numa única entidade, esta tem o poder absoluto sob os dados e as relações convergem no seu sentido, por outro lado, à direita a descentralização da informação permite garantir a integridade das transações, não existe uma entidade única no centro da rede e cada transação, entre os seus participantes, efetua-se por mecanismos de consenso (*consensus*) mútuo. Esta última apresenta um nível de segurança superior e oferece anonimato aos seus utilizadores (101 Blockchains, 2021).

Uma rede P2P consiste numa ligação direta através da *internet* ou *intranet* entre todos os participantes (*peers*), com igual distribuição de privilégios. A Figura 4 esquematiza uma rede P2P.

Figura 4 – Rede P2P (Brede, 2021).

Neste tipo de redes, todos os *peers* enviam/recebem pedidos e partilham os seus recursos computacionais, eliminando a necessidade de um servidor central, pelo que em caso de falha de um dos nós toda a rede não fica comprometida (Amandeep, 2020).

A Figura 5 exemplifica uma sequência de blocos com transações efetuadas numa BC.

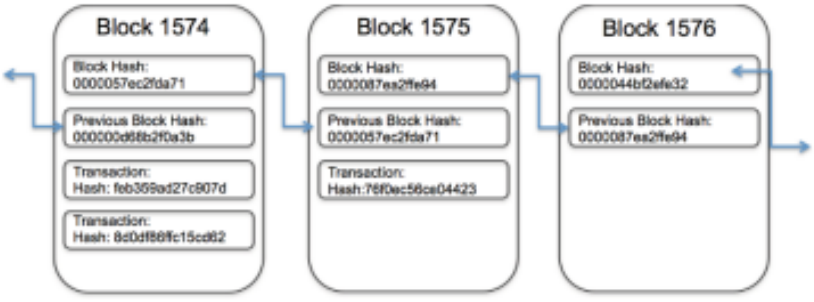


Figura 5 – Transações armazenadas em blocos de uma BC (Gupta, 2020).

As transações da BC são armazenadas em blocos, uma sequência de blocos (*blocks*) forma uma cadeia (*chain*).

Cada *block* é constituído por quatro *headers*:

- ***Previous Hash*****:** *Hash* do *block* anterior, criando desta forma uma ligação entre todos os *blocks;*
- ***Transaction Details*****:** Detalhes das transações que devem ocorrer;
- ***Nonce*****:** Número aleatório utilizado para diferenciar o endereço *hash* do *block*;
- ***Hash Address of the Block*****:** Endereço *hash* do *block*, gerado com base nos *headers* acima.

A tecnologia BC utiliza criptografia e funções de *hashing* para garantir segurança dos dados, recorrendo principalmente ao algoritmo SHA256 (Afreen, 2022).

2.1.1 Tipos de *Blockchain*

De acordo com os autores Shrivas & Yeboah, existem quatro tipos de *blockchains* (Shrivas & Yeboah, 2018):

- ***Blockchain*** **Pública:** Desenvolvidas para serem totalmente descentralizadas. Qualquer utilizador pode participar, efetuar novas transações e ver detalhes das transações já processadas. Possuem um *token* associado para incentivar e premiar os participantes, exemplos deste tipo de *chain* é a Bitcoin e Ethereum;
- ***Blockchain*** **Privada:** Os participantes precisam de consentimento para integrarem na rede, sendo que as transações estão apenas disponíveis internamente. São mais utilizadas para empresas que pretendem colaborar e partilhar dados, apresentando-se mais centralizadas que as *blockchains* públicas. Exemplos deste tipo são Hyperledger e R3 Corda;
- ***Blockchain*** **Híbrida:** Apresenta características das *blockchains* públicas e privadas. As transações são privadas e encontram-se no interior da rede, a autorização ao acesso dos dados pode ser gerida por participante. Exemplo deste tipo é a Dragonchain (Dragonchain, 2019);
- ***Blockchain*** **de Comunidade/Consórcio:** Contrariamente à *blockchain* privada, apresenta um modelo mais corporativo, gerido em grupo de entidades, e não apenas uma.

2.1.2 Mecanismos de *Consensus*

Como referido acima (secção 2.1), as transações entre os participantes da rede P2P dão-se com recurso a mecanismos de consenso mútuo (*mutual consensus*). Estes são algoritmos com a função de criar blocos na BC, de forma consensual entre os nós (participantes) da rede. Segundo os autores Aliaga & Henriques, existem diversos tipos de mecanismos de consenso, destacando-se os seguintes (Aliaga & Henriques, 2017):

- ***Proof-of-Work* (POW):** Consiste nos nós usarem o seu processamento computacional para solucionarem complexos desafios matemáticos, de forma a providenciar a solução e adicionar um novo *block* à *chain*. É o mecanismo utilizado pela Bitcoin;
- ***Proof-of-Stake* (POS):** O sistema faz a seleção do nó responsável por criar o *block*, com base na quantidade de *tokens* que o nó possui, ou seja, quanto maior a *"stake"* do nó, maior será a possibilidade de ser selecionado. Exemplo de BC que utiliza este mecanismo é a Cardano.

2.1.3 *Wallet* e *Token*

Uma carteira (*wallet*) BC permite aos utilizadores armazenar e gerir diferentes tipos de moedas digitais. É acessível através de dispositivos *web*, sem comprometer a identidade e privacidade do utilizador (Simplilearn, 2022).

O *token* é o termo utilizado em alternativa a criptomoeda ou criptoativo (Coinbase, 2022). Podem representar ativos ou utilitários fungíveis e negociáveis nas BC a que pertencem (Frankenfield & Mansa, 2021).

Cada *wallet* possui uma chave pública (*public key*) e uma chave privada (*private key*), sendo que a *public key* pode ser partilhada e funciona como um identificador na rede BC, enquanto a *private key* é secreta (Ledger, 2019).

A Tabela 1 apresenta os diversos tipos de *Wallet* existentes, segundo o autor Toshendra Sharma (Blockchain Council, 2022).

Tabela 1 – Tipos de BC *wallets*.

Tipo	Categoria	Nome	Descrição
Software	*Cold*	*Desktop Wallet*	A *private key* é armazenada localmente sendo possível efetuar transações *offline* e, posteriormente ficar *online*. Apenas são acessíveis no *hardware* onde se encontra instalada.
	Hot	*Online Wallet*	São executadas na *internet* e acessíveis através de um *browser*. A *private key* é armazenada *online* e gerida por terceiros.
		Mobile Wallet	Similares às *online wallet*, no entanto são desenvolvidas para o acesso em dispositivos móveis.
Hardware	*Cold*	-	Por exemplo, um dispositivo USB que armazena a *private key* da *wallet*. É necessário conectar o *hardware* no computador para aceder e efetuar transações.
Paper	*Cold*	-	Endereços da *wallet* são guardados fisicamente, num papel, por exemplo, um código QR impresso.

A necessidade da *wallet* se conectar à *internet* define a sua categoria, as *hot wallet* funcionam *online*, enquanto as *cold wallet* funcionam totalmente *offline*, apresentando por isso um nível superior de segurança.

2.1.4 Vantagens e Desvantagens do *Blockchain*

Como descrito nas secções anteriores, a tecnologia BC apresenta diversas características que se traduzem em vantagens face aos tradicionais sistemas centralizados. A imutabilidade e criptografia das transações, o fator descentralização e o elevado nível de segurança. De salientar a rápida execução das transações na BC (Afreen, 2022) (Essex, 2021).

Por outro lado, exibe algumas desvantagens, como o elevado consumo de tempo e energia elétrica pelos participantes na verificação dos *blocks* de transações, através do mecanismo de *consensus* POW.

A escalabilidade da BC apresenta limitações, para manter os níveis de confiança da rede as transações são processadas pelos participantes. Por exemplo, na BC *Bitcoin* o tamanho de cada *block* está limitado a um megabyte, o que restringe a quantidade de transações em cada *block* e exige constante criação de novos *blocks* (Geroni, 2021).

Por último, destaca-se a imaturidade, tratando-se de uma tecnologia recente e complexa, muitas pessoas não têm confiança na sua utilização (GeeksforGeeks, 2022) (Romanovs & Strebko, 2018).

2.2 Smart Contracts

Com base na secção inicial do enquadramento e contexto (secção 1.1), os *Smart Contracts* (SC) consistem na tradução de acordos entre participantes da BC em código possível de se executar, são contratos programáveis. A sua estrutura consiste na condição *"if… then"*, se determinada condição se verificar, então será despoletada uma ação.

A Figura 6 enquadra a utilização de *smart contracts* numa BC.

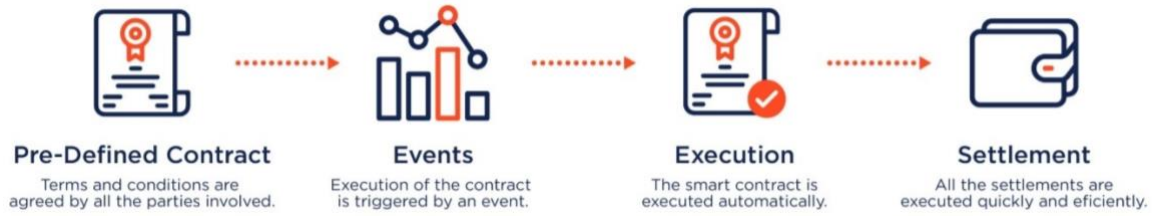


Figura 6 – Processo dos *smart contracts* na BC (DCX Learn, 2022).

A primeira fase da criação de um SC é a definição dos termos e condições entre todos os participantes envolvidos na BC. Uma vez definidos e alojados (*deployed*), estes reagirão a determinados eventos, iniciando a sua execução automaticamente a fim de realizarem as instruções que contêm programados.

Os SC apresentam um conjunto de vantagens na sua utilização, são eficientes, rápidos e precisos, pois a sua execução inicia sempre que determinadas condições são verificadas, eliminando a necessidade de efetuar contratos escritos manualmente, sujeitos a erros. A confiança, transparência e segurança são também características favoráveis dos SC, tal como o reduzido custo e tempo de execução (Arora, 2022).

Contudo, uma vez que estes encontram-se *deployed* na BC, poderá haver a necessidade de fornecer dados externos (*off-chain data*), para maximizar a sua utilidade. Para tal é indispensável uma ligação entre as fontes de dados externas e a BC e, aqui, surgem os oráculos (*oracles*) que providenciam uma camada entre os dados da BC (*on-chain*) as fontes de dados *off-chain* (Mou, 2020). O oráculo tem a capacidade de consultar, verificar e autenticar informação de fontes externas para o interior da BC, bem como, extrair informação para o exterior da BC, denominando-se oráculo *inbound* e oráculo *outbound*, respetivamente.

Pode-se categorizar os oráculos conforme o tipo de fonte de dados que utilizam, como oráculo de *software* ou *hardware*. Entende-se por oráculos de *software* os que utilizam fontes de informação digitais, como, base de dados, páginas *web* e servidores. Outrora, os oráculos de *hardware* podem recolher informação de sensores, termómetros, *scanners*, entre outros (Cryptopedia Staff, 2022).

2.3 Projetos Blockchain

Nesta secção pretende-se apresentar diferentes projetos tecnológicos existentes para o desenvolvimento de uma BC.

Posteriormente, no capítulo seguinte (secção 3.3), é justificada a opção tomada relativamente ao projeto escolhido para a elaboração do POC.

2.3.1 R3 - Corda

R3[1] é uma empresa de tecnologia e serviços corporativos que visam promover a colaboração digital com base na confiança, desenvolvendo DLT's privadas, seguras e escaláveis (R3, 2022). Encontra-se incluída num consórcio constituído por mais de 80 instituições internacionais, incluindo entidades como Bank of America, BBVA e Intel (Crosman, 2017).

O Corda[2] é um produto BC da R3, foi criado com a visão de um sistema capaz gerir e automatizar diferentes contratos legais, e com a convicção que o mercado precisaria de um modelo de negócios alternativo em que várias empresas colaboram para manter mutualismo. Consiste numa DLT, onde a implementação da lógica de negócio assenta em *smart contracts* e a sua arquitetura tem como *standards* a escalabilidade, longevidade, segurança, estabilidade, e interoperabilidade.

R3 dispõe de uma versão da tecnologia direcionada ao uso corporativo, denominado Corda Enterprise, que possibilita a integração com a versão base. Este produto pode ser utilizado em diversas áreas, como, a saúde, o governo, as finanças, seguros e, governo (Anwar, 2019).

2.3.2 Hyperledger Foundation

A Hyperledger Foundation[3] é uma organização sem fins lucrativos que pertence à Linux Foundation[4], consiste numa infraestrutura *open-source* capaz de fornecer ferramentas, *frameworks*, *standards* e bibliotecas, para o desenvolvimento de projetos em BC (Hyperledger Foundation, 2022).

O projeto Hyperledger foi lançado em 2016 e atualmente apresenta um conjunto numeroso de produtos disponíveis, destacando-se o Hyperledger Fabric (HLF). Este produto foi desenvolvido em cooperação com a empresa IBM. Dispõe de uma arquitetura modular onde são definidos *roles* pelos participantes na rede, apresenta suporte para *smart contracts* e mecanismos de *consensus* configuráveis (Gillis, 2021).

[1] Página oficial R3: 'www.r3.com'.
[2] Página oficial Corda: 'www.corda.net'.
[3] Página oficial Hyperledger Foundation: 'www.hyperledger.org'.
[4] Página oficial Linux Foundation: 'linuxfoundation.org'.

Atualmente, a tecnologia apresenta diversos casos de uso na indústria, a empresa de retalho Walmart utiliza-a na sua cadeia de fornecimento de alimentos. A Honeywell Aerospace fabricante de componentes de aviação possui um *marketplace* de peças desenvolvido com o HLF (Hyperledger Foundation, 2022).

2.3.3 Ethereum

"*Ethereum builds on Bitcoin's innovation, with some big differences*", o projeto[5] foi anunciado em 2013 pelo seu fundador Vitalik Buterin e lançado oficialmente em 2015, sendo o Ether (ETC) a *cryptocurrency* oficial. Apresenta-se como a tecnologia BC capaz de providenciar uma linguagem de programação *Turing* completa, com uma vasta aplicabilidade (Ethereum, 2022). Os principais casos de uso são:

- Finanças Descentralizadas (DeFi);
- *Tokens* Não-Fungíveis (NFT);
- Organizações Autónomas Descentralizadas (DAOs).

A Figura 7 apresenta a evolução do número de transações efetuadas na BC Ethereum e o respetivo valor da sua moeda, entre o seu lançamento e o segundo semestre de 2021.

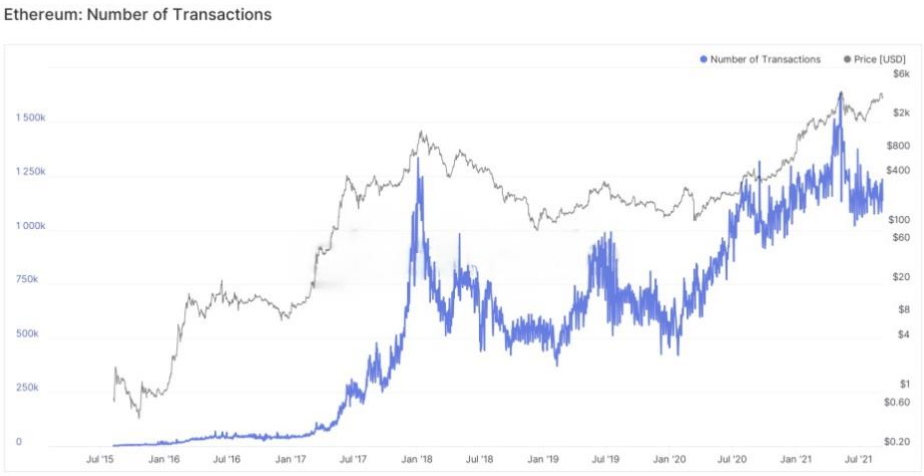


Figura 7 – Gráfico com transações efetuadas na BC Ethereum e, o valor do ETH, entre julho de 2015 e julho de 2021 (McNamara, 2021).

Pela figura anterior (Figura 7), é possível entender a evolução das transações efetuadas na BC Ethereum, estas apresentaram um crescimento acentuado durante o segundo ano da tecnologia no mercado, atingiu o seu máximo durante o primeiro semestre de 2021, sendo que o valor em *dollar* do ETH acompanhou o número crescente de transações.

[5] Página oficial Ethereum: 'ethereum.org'.

2.4 Utilização Empresarial Blockchain

Esta secção tem como objetivo descrever alguns casos de uso em que a tecnologia BC está atualmente a ser adotada como forma de inovação e alternativa aos sistemas tradicionais.

2.4.1 Walmart Canadá

A Walmart Inc.[6], fundada em 1962, é uma empresa multinacional americana da área do retalho que opera cadeias de hipermercados, mercearias e centros comerciais (Walmart, 2022).

Atualmente, a Walmart Canadá para operar com informações de faturas e pagamentos de cerca de setenta transportadoras parceiras, apresenta uma solução *blockchain* desenvolvida em colaboração com a empresa DLT Labs[7] (Srivastava, et al., 2022).

Esta solução surgiu da necessidade de resolver inconsistências de dados das faturas e pagamentos às transportadoras, originados pela incompatibilidade de comunicação entre os sistemas utilizados pela Walmart e as transportadoras. Estes problemas apresentavam impacto direto nos prazos de pagamento, impossibilitando o cumprimento dos mesmos e, como resultado disso, encargos financeiros para a empresa (Mukherjee, 2022).

A utilização da tecnologia *blockchain* apresentou diversas vantagens, como (Galea-Pace, 2020):

- DLT promove a confiabilidade e automatiza processos do negócio;
- Sistema mais eficiente, melhoria na gestão dos recursos, com repostas mais rápidas, antecipação de problemas e redução de custos;
- Transações imediatas permitem consolidar em tempo real as regras de negócio, reduzindo os tempos de espera e promovendo pagamentos mais rápidos;
- Dados precisos em tempo real permitem uma análise mais aprimorada, melhorando a capacidade de planeamento e orçamento.

O sistema desenvolvido teve um período de testes de dois anos e após entrada em operação imediatamente revelou enormes benefícios, reduzindo a taxa de faturas contestadas de 70% para apenas 1% (Apgar, 2022).

[6] Página oficial Walmart: 'corporate.walmart.com'.
[7] Página oficial DLT Labs: 'www.dltlabs.com'.

2.4.2 Etherisc

Etherisc[8] é uma plataforma descentralizada de aplicações da indústria dos seguros (*insurance*). Tem o objetivo de aumentar a eficiência e transparência na comercialização de seguros, diminuindo os custos operacionais, e promover a democratização no acesso a investimentos em resseguros, através da tecnologia *blockchain* (Hyperledger, 2022).

Atualmente apresenta diversos produtos, alguns ainda em fase de protótipo:

- Seguro agropecuário disponibiliza pagamentos automatizados, acionados por situações de seca ou inundação relatados por agências governamentais;
- Seguro de atraso de voo com pagamentos automáticos e instantâneos;
- Seguro de carteira de cripto moedas protege contra o risco de roubo e ataques de *hackers*;
- Proteção colateral para empréstimos em cripto, assegura o pagamento de todo o valor emprestado caso o valor do *token* desvalorize;
- Seguro social contra o risco de morte ou doença grave, com pagamento imediato.

Na base do funcionamento dos produtos da plataforma Etherisc está o conceito seguro paramétrico. Este conceito consiste em utilizar oráculos como fontes de dados para algoritmos de pagamentos e subscrições, ou seja, sempre que determinadas condições se verificam automaticamente são executados SC's que determinam possíveis transações monetárias.

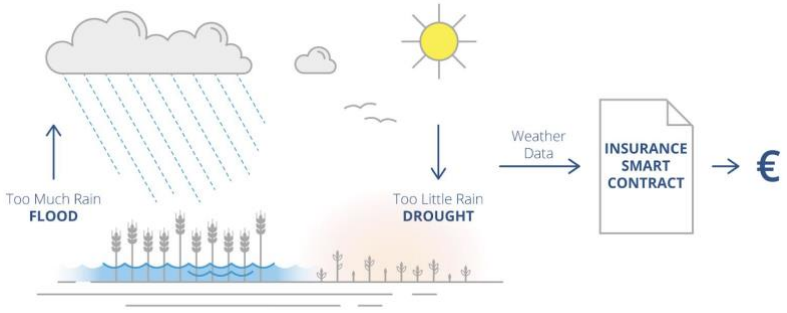


Figura 8 - Esquematização gráfica do funcionamento do seguro agropecuário paramétrico (Etherisc, 2022).

A Figura 8 apresenta uma esquematização do conceito seguro paramétrico para o seguro agropecuário acima mencionado, durante a ocorrência de fenómenos meteorológicos extremos, dados da meteorologia recolhidos pelos oráculos servem de *input* para SC's despoletarem as indenizações pré acordadas com os clientes.

[8] Página official Etherisc: 'etherisc.com'.

3 Análise de Valor

Neste capítulo, pretende-se apresentar a análise de valor da utilização da tecnologia BC e SC's no desenvolvimento de sistemas com diversos participantes. São apresentados os conceitos de valor, proposta de valor e valor percebido. De seguida, é descrito o processo de negócio e inovação, posteriormente, o processo de hierarquia analítica e, por fim, a análise funcional do problema.

A análise de valor permite validar se um determinado produto atende aos requisitos dos seus consumidores, com o menor custo possível, sem comprometer os padrões de performance e confiabilidade. É um processo formal, sistemático, e organizado, de análise e avaliação, direcionado à função e utilidade de um produto (Rich & Holweg, 2000).

3.1 Valor, Proposta de Valor e Valor Percebido

O conceito valor é impreciso e pode assumir vários significados dependendo do contexto. Do ponto de vista da engenharia, valor é um algo quantitativo e expresso na satisfação do cliente ou custo do produto. A equação 1 expressa a razão entre a funcionalidade e custo do produto (Rich & Holweg, 2000).

$$Valor = \frac{(Performance + Capacidade)}{Custo} = \frac{Função}{Custo} \quad (1)$$

A performance e capacidade estão diretamente relacionadas com as características do produto, resultando na sua função. O valor de um produto é mesurável através da razão entre a função e custo.

Proposta de valor (*value proposition*) pode ser desenvolvida com base em várias perspetivas, a do produto, negócio, e perspetiva do investidor. A perspetiva do produto consiste numa declaração que demostra a relação entre as necessidades dos consumidores e os benefícios resultantes da utilização do produto, de forma sucinta e credível, que atenda a uma necessidade do mercado (Andrade, 2022).

O Valor Percebido é a perceção que os consumidores têm face a um produto e qual o custo que estes estão dispostos a suportar pelo bem ou serviço, sendo que as características do produto poderão ser impulsionadas pelo marketing (Koop & Mansa, 2020).

3.2 Processo de Negócio e Inovação

O trabalho elaborado neste documento consiste num POC, que propõe um sistema inovador para um processo industrial, em alternativa aos sistemas tradicionais atualmente em operação.

O processo de inovação é constituído por três áreas distintas, o *Fuzzy Front End* (FFE), o *New Product Development* (NPD) e, Comercialização (*Commercialization*). A Figura 9 enquadra estas três áreas do processo.

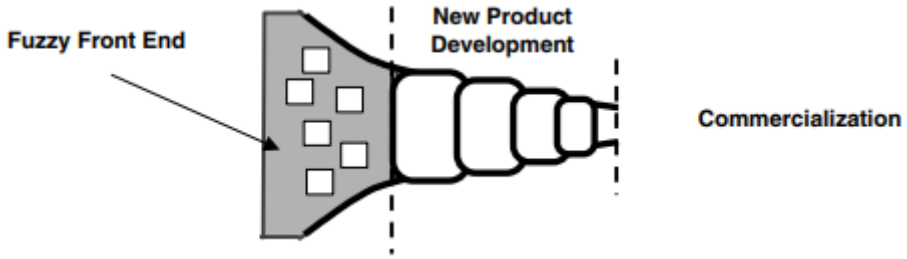


Figura 9 – As três partes constituintes do processo de inovação: FFE, NPD e Comercialização.

A Tabela 2 apresenta uma comparação entre as etapas FFE e NPD do processo de inovação (Koen, et al., 2004).

Tabela 2 – Diferenças entre o FFE e NPD.

Fator	FFE	NPD
Natureza do trabalho	Experimental	Disciplinada e orientada ao objetivo
Data de comercialização	Incerta/Imprevisível	Alto grau de certeza
Financiamento	Variável	Orçamento
Expectativas de retorno	Incerta (especulativa)	Previsível
Atividade	Individual	Produto multifuncional
Medidas de progresso	Conceitos justificados	Conclusão de *milestones*

A Tabela 3 apresenta os diversos elementos do NPD, aplicados no POC desenvolvido neste documento de dissertação (Koen, et al., 2004).

Tabela 3 – Elementos constituintes do NPD.

Identificação da oportunidade	Exploração e aplicação da área de BC e SM's em processos de negócios (descrito na secção 1.1).
Análise da oportunidade	Apresentar alternativas aos sistemas tradicionalmente utilizados nos diversos setores industriais.
Gênese da ideia	Obter conhecimento na área BC e na sua aplicabilidade.
Seleção da ideia	Expandir o atual mercado do aluguer de automóveis e promover relações de mutualismo entre diversas empresas da mesma área.
Definição do conceito	POC descrito neste documento.

3.3 Processo de Hierarquia Analítica (AHP)

Nesta secção, descreve-se a aplicação de um método de análise hierárquica. O objetivo final é definir o tipo sistema para o desenvolvimento do POC proposto neste documento de dissertação e, por conseguinte, determinar a tecnologia a utilizar.

Analytic Hierarchy Process (AHP) é uma técnica de decisão multicritério, que permite utilizar vários critérios qualitativos e/ou quantitativos. Consiste em segmentar o problema de decisão em níveis hierárquicos, no sentido de encontrar a melhor alternativa (Araujo, 2022).

A aplicação do método AHP divide-se em quatro etapas (Lapevski & Timovski, 2014):

1. Definição do problema;
2. Desenvolvimento de um modelo hierárquico, na qual é apresentado o problema no topo, de seguida os critérios e no final as alternativas;
3. Avaliação da importância relativa de cada um dos níveis do modelo hierárquico;
4. Seleção da alternativa com melhor prioridade global, com base nas prioridades obtidas pelas comparações dos diferentes níveis hierárquicos.

3.3.1 Tipo de Sistema

Para definir o tipo de sistema no desenvolvimento do POC são definidos os seguintes critérios:

- **Custo:** custo desenvolvimento e manutenção do sistema;
- **Segurança:** segurança que o sistema garante nas suas transações;
- **Automatização:** automatização de interações entre parceiros de negócio com acesso ao sistema.

As alternativas para o tipo de sistema são:

- **Descentralizado:** rede BC com utilização de SC's, sem entidade centralizadora;
- **Centralizado:** sistema com entidade centralizadora dos dados;
- **Distribuído:** sistema com dados distribuídos pelos nós.

A Figura 10 representa o diagrama hierárquico do método AHP.

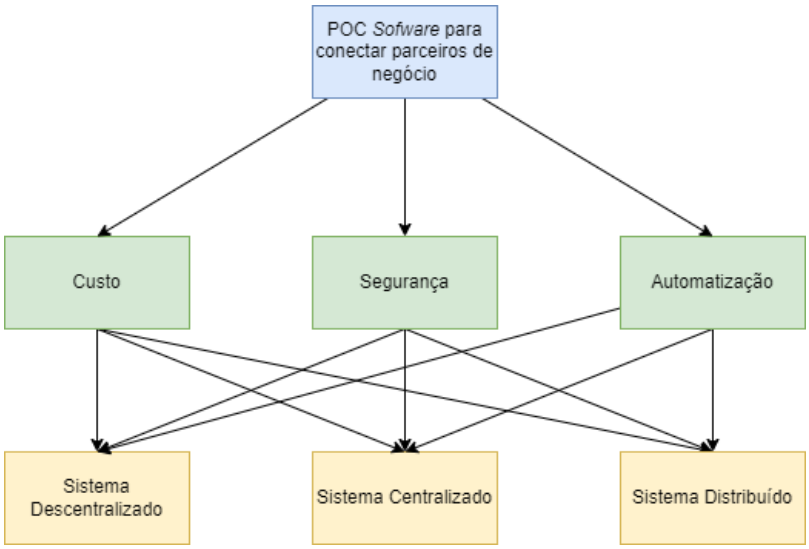


Figura 10 – Árvore de decisão hierárquica com a relação entre os critérios e alternativas.

A Tabela 4 contém a escala do AHP, utilizada na atribuição de importância dos critérios e alternativas.

Tabela 4 – Escala com níveis de importância a atribuir aos critérios (Saaty, 1987).

Nível de Importância	Definição
1	Igual importância
3	Fraca importância
5	Forte importância
7	Importância muito forte
9	Importância absoluta
2 – 4 – 6 - 8	Condição intermédia entre níveis

A Tabela 5 contém a atribuição de importância dos vários critérios considerados.

Tabela 5 – Níveis de importância AHP dos critérios.

Critério	Custo	Segurança	Automatização
Custo	1	1/3	1/7
Segurança	3	1	1/4
Automatização	7	4	1

A Tabela 6 apresenta a importância atribuída às diferentes alternativas, para o critério custo.

Tabela 6 - Níveis de importância AHP do critério custo.

Custo	Sistema Descentralizado	Sistema Centralizado	Sistema Distribuído
Sistema Descentralizado	1	6	2
Sistema Centralizado	1/6	1	1/4
Sistema Distribuído	1/2	4	1

A Tabela 7 apresenta a importância atribuída às diferentes alternativas, para o critério segurança.

Tabela 7 - Níveis de importância AHP do critério segurança.

Segurança	Sistema Descentralizado	Sistema Centralizado	Sistema Distribuído
Sistema Descentralizado	1	6	5
Sistema Centralizado	1/6	1	1/4
Sistema Distribuído	1/5	4	1

A Tabela 8 apresenta para o critério automatização com a importância distribuída pelas alternativas.

Tabela 8 - Níveis de importância AHP do critério automatização.

Automatização	Sistema Descentralizado	Sistema Centralizado	Sistema Distribuído
Sistema Descentralizado	1	8	5
Sistema Centralizado	1/8	1	1/3
Sistema Distribuído	1/5	3	1

Por fim, Tabela 9 mostra os resultados obtidos da aplicação do método AHP.

Tabela 9 – Resultados da aplicação do método AHP.

Tipo de Sistema/Critério	Custo	Segurança	Automatização	Resultado
Sistema Descentralizado	0.61	0.69	0.74	0.63
Sistema Centralizado	0.06	0.08	0.08	0.06
Sistema Distribuído	0.33	0.22	0.19	0.30

De acordo com a tabela acima (Tabela 9), o sistema descentralizado apresenta o melhor resultado.

3.3.2 Projeto Blockchain

Atendendo ao resultado obtido anteriorment (secção 3.3.1), para determinar o projeto BC a utilizar no desenvolvimento do POC, apresenta-se a seguinte Tabela 10, que com base nas características descritas na secção Descrição do Problema (secção 1.2), compara as tecnologias mencionadas no capítulo Estado da Arte (secção 2.3). Conclui-se que o projeto BC a utilizar no desenvolvimento do trabalho descrito neste documento de dissertação é o Hyperledger Fabric (HLF).

Tabela 10 - Comparação entre Ethereum, R3 – Corda e Hyperledger Fabric (Derecha, 2023) (Abrol, 2022) (Ethereum, 2023).

Aspeto Comparativo	Ethereum	R3 - Corda	Hyperledger Fabric
Tipo	- Pública.	- Privada.	- Privada.
Casos de Uso	- Fornece uma *framework* para o desenvolvimento de aplicações descentralizadas; - Execução numa rede P2P.	- Plataforma DLT personalizada para a indústria financeira.	- Tecnologia *open-source* para soluções de BC industriais (e.g. *supply chain*); - Visa acelerar do desenvolvimento da tecnologia BC entre vários setores industriais.
Smart Contracts	- Suporta o desenvolvimento de SM's, na linguagem de programação Solidity.	- Suporta o desenvolvimento de SM's, na linguagem de programação Kotlin.	- Suporta desenvolvimento de SM's, nas linguagens de programação Golang, Java e JavaScript (com NodeJs).
Autenticação/ Autorização	- *Descentralized Identity*, os identificadores descentralizados (DIDs) não são geridos por nenhuma entidade; - Infraestrutura baseada em chaves publicas (PKI) e privadas, com recurso a bases de dados descentralizadas; - Os identificadores são validados na BC através da chave pública.	- *Certificate authority* com base numa PKI e certificados X.509; - *Nodes* necessitam solicitar acesso à rede, sendo o serviço de autenticação responsável por identificar os *nodes* (KYC); - Chaves privadas dos participantes armazenadas num nó.	- *Certificate authority* com base numa PKI e certificados X.509; - Todas as organizações possuem o seu próprio CA; - Customização de certificados X.509 para especificar políticas de autorização (secção 4.1).
Performance (TPS)	- Entre 15 e 20 TPS (Agarwal, et al., 2022).	- Entre 15 e cerca de 1700 TPS (Corda, 2018).	- Aproximadamente 3500 TPS (Agarwal, et al., 2022); - Possibilidade de otimização arquitetural para atingir as 20000 TPS (Gorenflo, et al., 2019).
Consensus	- Algoritmo *Proof-of-Stake* (secção 2.1.2).	- *Validity* e *Uniqueness consensus* (R3, 2023).	- Fluxo transacional: *Endorsement*, *Ordering* e *Validation* (secção 4.5).

3.4 Proposta de Valor

Tendo por base a definição de proposta de valor descrita na secção acima (secção 3.1), com recurso à ferramenta *Value Proposition Canvas* é possível posicionar o valor do produto e as necessidades dos consumidores.

A Figura 11 apresenta o *Value Proposition Canvas* para o trabalho desenvolvido neste documento de dissertação. Esta proposta divide-se em duas partes, à direita o perfil de cliente (*cliente profile*) subdivido entre os trabalhos a fazer (*jobs-to-be-done*), os ganhos (*gains*) e esforços (*pains*). À esquerda os produtos e serviços (*products & services*), constituído pelo ganho dos criadores (*gain creators*) e "analgésicos" (*pain relievers*).

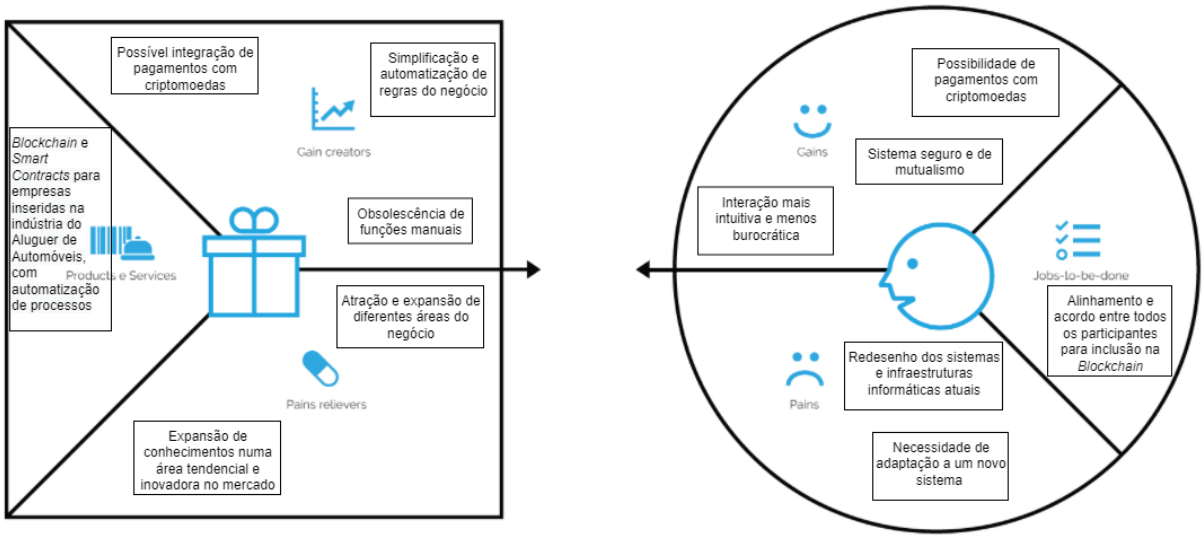


Figura 11 – *Value Proposition Canvas* proposto para o produto (adaptado (Pereira, 2021)).

A proposta de valor do trabalho desenvolvido neste documento visa sobretudo:

- **Desenvolvimento de Conhecimento:** a área do BC é relativamente recente no mercado e persiste um desconhecimento generalizado do potencial e utilidade da tecnologia entre a comunidade (desenvolvedores e empresas). Este trabalho pretende criar conhecimentos na área do BC e SC's, de modo a demonstrar a aplicabilidade em negócios do quotidiano.
- **Automatização de Processos de Negócio:** tendencialmente os sistemas informáticos tendem a crescer e ficarem mais complexos, devido as crescentes exigências dos negócios. A procura pela automatização de interações é indispensável e simultaneamente complexa, no entanto a tecnologia BC pode simplificar muitas interações entre participantes, e os SC's garantirem a integridade das transações envolvidas no negócio, criando-se assim um sistema de mutualismo entre todos os parceiros.

3.4.1 Blockchain e Smart Contracts – Vantagens e Desvantagens

Como já descrito em secções anteriores (secção 2.1 e 2.2), a implementação de sistemas com BC e SC's revela vantagens e desvantagens da sua utilização.

A Tabela 11 descreve diversas vantagens e desvantagens da utilização de BC e SM's.

Tabela 11 – Vantagens e desvantagens do BC e SM's.

Conceito	Vantagens	Desvantagens
Blockchain	- Sistema descentralizado - Transações imutáveis e criptografadas - Segurança - Performance	- Alto consumo de energia para validar transações (para determinados mecanismos de *consensus*) - Escalabilidade complexa; - Desconhecimento geral da comunidade
Smart Contracts	- Eficiência e rapidez - Autonomia - Confidencialidade e transparência	- Baixa popularidade - Escassez de mão de obra

3.5 Análise Funcional do Problema

A análise funcional de um produto é determinante e pode ser desenvolvida com recurso a diversas técnicas. A *Function Analysis System Technique* (FAST) consiste numa representação gráfica que mostra as relações lógicas entre as funções de um produto, com base nas questões "Como/Porquê/Quando?" ("*How/Why/When?*") em relação ao *scope* do produto (Value Analysis Canada, 2021).

A Figura 12 apresenta um diagrama FAST, trata-se da análise funcional do problema descrito para atender ao POC descrito neste documento de dissertação, é um processo criativo que auxilía na compreensão do problema.

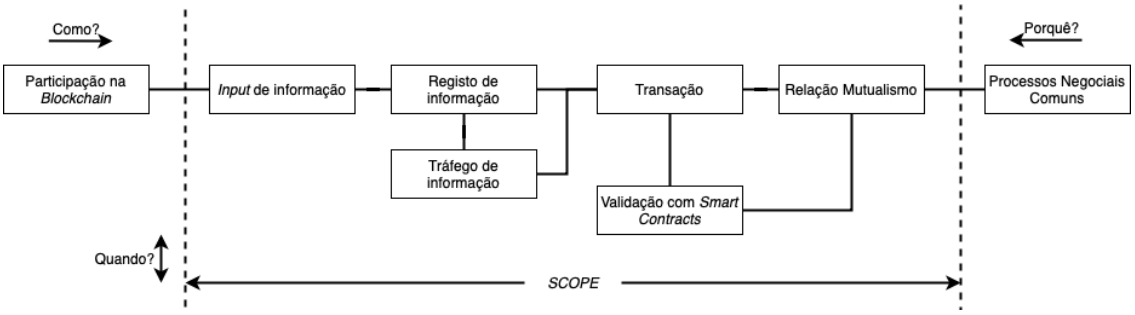


Figura 12 – Diagrama FAST representativa da análise funcional do problema (adaptado (Araujo, 2022)).

A participação das empresas de aluguer automóvel, clientes e outras entidades na BC pressupõe o tráfego de informação como resultado das interações do negócio. O *input* de informação origina um conjunto de reações no sistema, estas informações são registadas na BC sob a forma de transações, sendo os SC's responsáveis pela sua validação, de forma a garantir a relação de mutualismo entre todos os participantes.

4 Hyperledger Fabric

O Hyperledger Fabric (HLF) é uma plataforma DLT com vasta aplicabilidade nos diversos setores industriais, em função da sua versatilidade e modularidade, garantida pelos componentes *plug and play*, mecanismo de consenso e serviços de autenticação/autorização (Hyperledger, 2023).

Este capítulo pretende explicar os principais conceitos da tecnologia HLF, anteriormente mencionada nas secções 2.3.2 e 3.3.2, com o objetivo de promover uma compreensiva interpretação dos seguintes capítulos.

Deste modo, o capítulo inicia-se pela descrição dos certificados de autorização (*certificates authorities*) (CA), os serviços de autenticação/autorização e políticas (policies) requeridas para operar na BC, seguindo-se com a definição dos principais componentes, os *peers* e *ledger*. Por último enquadra-se o conceito de SM's no HLF, denominados *chaincodes*.

4.1 MSP e CA

O Membership Service Provider (MSP) e Fabric Certificate Authorities (CA) são componentes essenciais no funcionamento do HLF. Esta tecnologia consiste numa rede com autorizações de acesso, ou seja, tratando-se de uma BC *permissioned*, possui nativamente estes componentes, um é o gestor das identidades (*identities*) internas e externas que acedem aos serviços da BC enquanto que o outro tem a responsabilidade da criação e emissão dos certificados de acesso, respetivamente.

Todos os componentes participantes na rede necessitam de uma *digital identity*, trata-se de um documento digital onde constam informações do seu titular, por exemplo, as permissões e tipos de acesso aos dados na BC.

O Fabric CA utiliza certificados do *standard* X.509, é responsável por assinar e emitir certificados válidos pelos participantes, ilustrado abaixo na Figura 13.

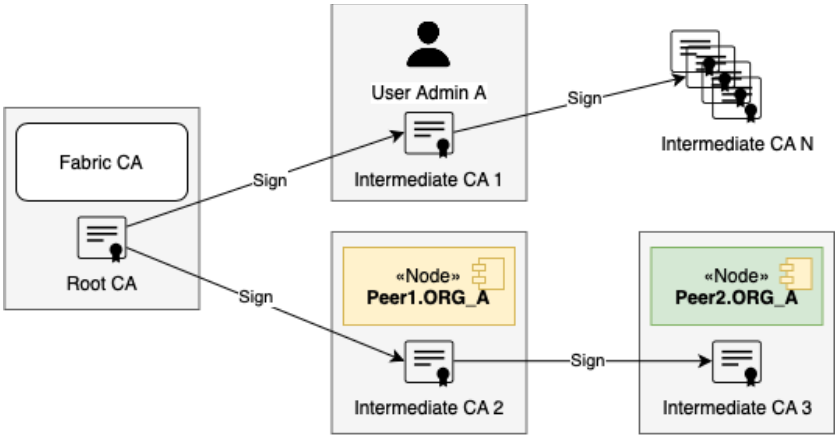


Figura 13 – Exemplo ilustrativo da atribuição de certificados CA no HLF (Hyperledger, 2020).

Com base na Figura 13, poderão existir dois tipos de certificados, CA's raiz e intermediários, sendo que ambos possuem um par *public key* e *private key*. Os certificados intermediários ("*Intermediate CA 1*" e "*Intermediate CA 2*") têm origem em certificados raiz ("*Root CA*"), e podem emitir outros certificados intermediários ("*Intermediate CA 3*" e "Intermediate *CA N*"), o que permite estabelecer cadeias de confiança nos participantes da rede (Hyperledger, 2020).

Por sua vez, o MSP é determinante a garantir a fidelidade do CA's, apresenta mecanismos que com base na *public key* validam a assinatura de uma transação. Tem a capacidade de identificar os privilégios específicos de uma *identity*, de forma a determinar que ações cada componente da organização pode desempenhar. Apesar do objetivo de funcionamento ser o mesmo, o MSP apresenta dois tipos, classificativos do nível (*scope*) em que operam:

- ***Local:*** utilizados em *clients* e *nodes* (*peers* e *orderer*), sendo que cada componente deve ter o seu MSP localmente configurado, pois este define as suas permissões. Possibilita a autenticação com o componente membro de um *channel;*
- ***Channel:*** define acessos administrativos e participativos ao nível do *channel*, ou seja, caso uma organização pretenda participar no *channel* deverá ser adicionado um MSP desta na configuração no mesmo. O MSP *channel* é instanciado no sistema de ficheiros de cada componente, mantendo a sua sincronização por *consensus*. Este também inclui os MSP's de todas as organizações ligadas ao *channel* (Hyperledger, 2020).

A sua configuração é efetuada num diretório "msp" em cada componente, com a estrutura de diretórios apresentada na Tabela 12.

Tabela 12 - Descrição dos vários diretórios constituintes da configuração do MSP (Hyperledger, 2020).

Diretório	Descrição
cacerts	Certificados X.509 *root*, da organização representada pelo MSP. Todos os outros certificados deverão ser originados pelos presentes neste diretório, de forma a preservar a cadeia de confiança na BC.
intermediatecerts	Certificados X.509 *intermediate*, da organização representada pelo MSP. Caso existam, foram originados com base nos certificados do diretório "cacerts".
admincerts	Certificados X.509 das *Identities* que definem os atores com *role* de administrador da organização.
keystore	Contém a *private key* um *node*, utilizada para assinar dados (e.g. transações). Diretório apenas utilizado para *Local* MSP.
signcert	Certificado do *node* representativo da sua *identity*, incluindo a *public key*.
tlscerts	Certificados X.509, emitidos pelos certificados *root* da organização representada pelo MSP e utlizados para efetuar comunicações seguras com o protocolo TLS.
tlsintermediatecacerts	Certificados X.509 *intermediate*, emitidos pelos certificados *root* da organização representada pelo MSP e utlizados para efetuar comunicações seguras entre os *nodes* com o protocolo TLS.

Note-se, na Tabela 12, que o conteúdo dos diretórios "*tlscerts"* e *"tlsintermediatecacerts"* apenas é justificável aquando do uso do protocolo TLS.

4.2 Policies

Policies (políticas) são um conjunto de regras, procedimentos e princípios que definem claramente a forma como determinadas ações são tomadas. Geralmente, indicam "quem" tem o direito de aceder "ao quê", por exemplo, informações ou ativos (Point, 2023).

Considerando o conceito de *policies* no contexto do HLF, definem como os participantes da BC acordam mudanças na mesma, por exemplo, alterações aos *channels* e *chaincodes*. Aquando da configuração ou modificação de um *channel*, os participantes da BC devem aceitar as *policies*, pois são o mecanismo da tecnologia que gere a infraestrutura.

Tal como anteriormente descrito (secção 4.1), as BC's em HLF são redes *permissioned* onde todos os componentes necessitam de um MSP, a definição de *policies* é determinante para uma administração harmoniosa numa rede em execução (Hyperledger, 2020).

No HLF as *policies* classificam-se em três tipos:

- ***Access Control Lists* (ACL):** configuração do controlo de acesso a recursos da BC, por exemplo, invocação de funções implementadas nos chaincodes. É principalmente utilizada pelos administradores da BC;
- ***Smart Contract Endorsement*:** especifica quantos participantes no *channel* precisam de executar e validar uma transação num determinado *smart contract*;
- ***Modification*:** determina as *identities* necessárias para aprovar qualquer alteração na configuração, por exemplo, de outras *policies*.

4.3 Peer e Channel

Os nós *peer* são componentes fundamentais de uma rede desenvolvida no HLF, pois podem hospedar cópias dos *ledgers* (secção 4.4) e *chaincodes* (secção 4.5), tornando-se assim o ponto de contacto entre uma aplicação (*application*) e os recursos da rede (Hyperledger, 2020).

A Figura 14 apresenta uma visão geral da interação estabelecida entre uma aplicação e um *peer*, incluindo os recursos a este associado, *ledgers, chaincode* e *orderer*.

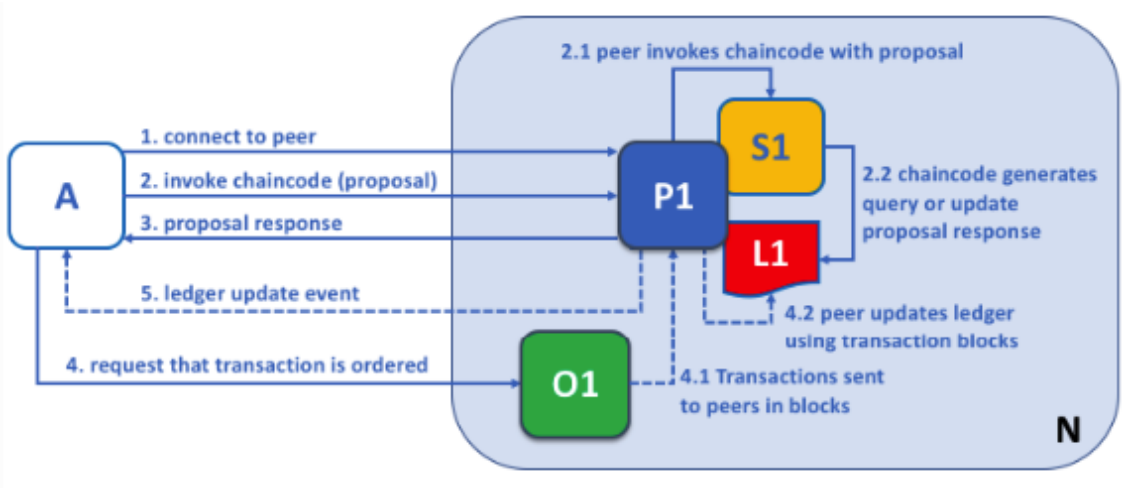


Figura 14 – Vista lógica da interação entre uma aplicação e um *peer* da rede HLF (Hyperledger, 2020).

Partindo da figura acima (Figura 14), as aplicações (A) conectam-se sempre diretamente aos *peers* (P1), de modo a consultar a informação no *ledger* (L1) e/ou executar os *chaincodes* (S1) (passos 1, 2, 2.1 e 2.2). Os pedidos da aplicação (A) feitas ao *peer* (P1) geram uma resposta não definitiva, a proposta (*proposal*) (passo 3) com o resultado da leitura/alteração ao *ledger* (L1). Posteriormente, a aplicação (A) submete uma transação para o *orderer* (O1), este criará os blocos de transações e distribui pelos *peers* disponíveis na rede (N), incluindo o *peer* P1 (passos 4, 4.1 e 4.2). Por fim, o *peer* valida a transação, atualiza o seu *ledger* (L1) e retorna para a aplicação (A) um evento notificativo da conclusão da transação.

Com vista numa melhor flexibilidade do *design* das redes HLF, é possível definir vários *peers* para uma organização, bem como, hospedar múltiplos *ledgers* e múltiplos *chaincodes* num único *peer*, o que aumenta a redundância do sistema e evita os pontos de falha única.

Os *peers* em funcionamento na rede poderão apresentar funções distintas, estabelecidas pelo seu tipo. Consideram-se os seguintes tipos de *peer*:

- ***Regular:*** nó *peer* constituinte da rede;
- ***Anchor:*** permite efetuar ligação entre diferentes organizações, pois é conhecido externamente à organização a que pertence. Apenas os *anchor peers* têm a capacidade de conexão a *anchor peers* de outra organização, utilizando o protocolo Gossip[9];
- ***Leader:*** recebe os novos blocos de transações enviados pelo *orderer*, e dissemina-os pelos outros *peers* da organização;
- ***Endorsing:*** participa no processo de assinatura das propostas de transação, de forma que esta seja considerada válida conforme definido na *endorsment policy* (secção 4.2) (Sakhuja, 2018).

O conceito *channel* no HLF é essencial para adicionar privacidade e confidencialidade às transações entre várias organizações, uma vez que são definidos pelos participantes e delimitam os *peers* e informação na rede (Oracle, 2023). A seguinte figura (Figura 15) exemplifica uma rede HLF com várias organizações ligadas a um *channel*.

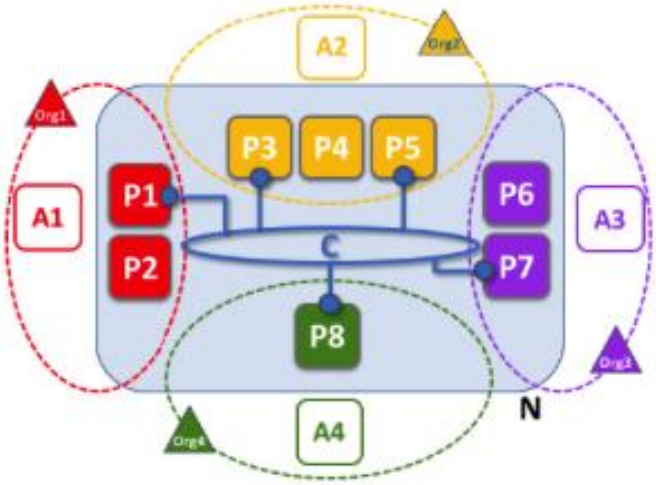


Figura 15 - Vista lógica dos componentes *peers* e *channel* numa rede HLF, segmentados por várias organizações (Hyperledger, 2020).

A rede N incorpora quatro organizações (Org1, Org2, Org3 e Org4), cada uma com uma única aplicação (A1, A2, A3 e A4), mas com número variável de *peers*. Por exemplo, a Org1 apresenta dois *peers* onde apenas o P1 está diretamente ligado ao *channel* (C), enquanto a Org2 apresenta dois *peers* (P3 e P5) ligados ao *channel* (C). De notar que seria possível definir outro *channel* para incluir *peers* das organizações acima e *peers* de outras novas organizações, restringindo-se os participantes com acesso à rede.

[9] Gossip é um protocolo de comunicação que permite a partilha do *state* entre nós de uma rede/cluster (Abbas, 2023).

Previamente nesta secção, introduziu-se o conceito *orderer* na comunicação entre uma aplicação e um *peer* da rede. O *orderer* é o componente que garante a ordem e consistência dos *ledgers* dos *peers* no decorrer das interações entre os *peers* e aplicações. Na sua implementação o HLF suporta diversas tecnologias, o Apache Kafka[10], Raft e nativamente o Solo.

No HLF entende-se dois tipos de transações, as de leitura e de escrita, sendo que estas últimas requerem o consentimento de outros *peers* para que uma atualização do *ledger* possa realmente ser efetuada. Posteriormente, atingido o *consensus* (secção 2.1.2), todos os *ledgers* dos *peers* são devidamente atualizados para o estado mais recente e as aplicações notificadas (Hyperledger, 2020). A Tabela 13 descreve as fases do fluxo transacional inerentes ao processo que garante a consistência dos *ledgers* em todos os componentes da rede no fluxo de uma transação.

Tabela 13 – Fases do fluxo transacional do HLF (Hyperledger, 2020).

Fase	Designação	Descrição
1	*Proposal*	- Fase inicial, a primeira interação entre a aplicação e o *peer* (não envolve o *orderer*). A aplicação submete propostas (*proposals*) de transações para os *peers* (os definidos para o *endorsement*); - Os *peers* assinam digitalmente as *proposals* e devolvem uma resposta para a aplicação, sem alterar o estado dos seus *ledgers*; - Assim que aplicação receber o número suficiente de respostas dos *peers*, pode transitar para a fase 2; - Note-se que a execução de *chaincodes* é não determinística, pois para uma transação os *ledgers* dos *peers* poderão ter estados distintos, no instante de execução dos mesmos.
2	*Ordering and packaging transactions into blocks*	- O *orderer* recebe as transações submetidas pela aplicação (as *proposals*), que previamente tinham sido *endorsered* pelos *peers* (na fase 1), e cria os blocos de transações para serem distribuídos pelos *peers*; - A ordem das transações no bloco é definida pelo *orderer*, e esta será a mesma utilizada para a execução da fase 3.
3	*Validation and Commit*	- Os blocos "*packaged"* pelo *orderer* na fase 2, são distribuídos pelos *peers* (não necessariamente todos); - Os *peers* que recebem os blocos procedem à validação final de todas as transações neles contidas, de forma independente, isto é, cada *peer* valida individualmente os blocos de forma determinística; - A validação (*validation*) é executada com base na *endorsement policy* (secção 4.2) definida pelas organizações; - Consumada a *validation*, os *ledgers* podem ser atualizados, ocorre o *commit*; - Note-se que transações falhadas são mantidas no bloco original para efeitos de auditoria.

[10] Página oficial Apache Kafka: kafka.apache.org/intro

4.4 Ledger

Um *ledger* é uma base de dados que regista cronologicamente transações (Natarajan, et al., 2017). Sendo o HFL baseado numa DLT, o controlo sobre o *ledger* existe nos vários nós constituintes da rede.

No HLF o *ledger* consiste em duas partes distintas, como ilustrado na Figura 16 (Hyperledger, 2020):

- ***World State*:** é uma base de dados com o estado mais atualizado dos mesmos, geralmente baseado num conjunto de pares *key-value* e que pode ser alterado, ou seja, é mutável. Utilizado para facilitar as consultas, eliminando a necessidade de processar o histórico de transações.
- ***Blockchain*:** tal como já mencionado anteriormente (secção 2.1), trata-se do histórico (*log*) de transações que resultaram no *world state*, é imutável e organiza-se em blocos.

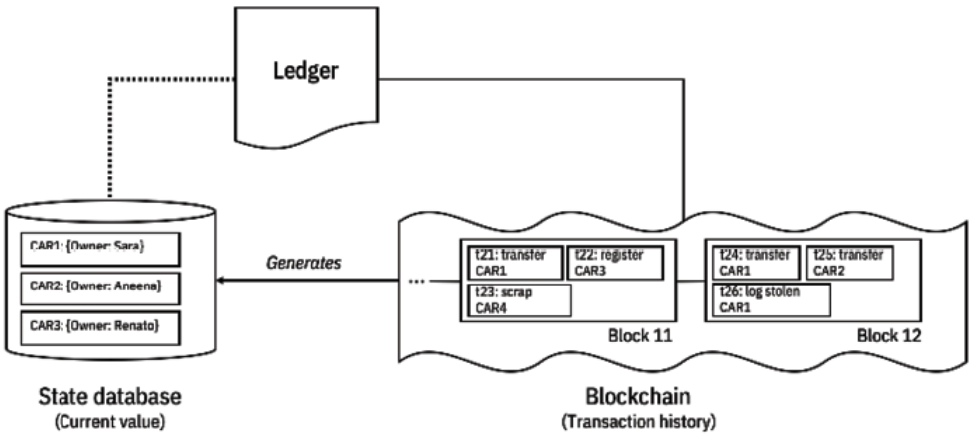


Figura 16 – Constituição do *ledger*, com *world state* e *blockchain* (Packt, 2023).

Pela figura acima (Figura 16), o *ledger* é a compreensão do *state database* (ou *world state*) e *blockchain*, isto é, a *blockchain* determina o estado final do *state database*. Note-se que esta estrutura no HLF é replicada de forma consistente em vários componentes da rede.

Na implementação do *ledger* no HLF existem duas possibilidades, LevelDB e CouchDB. É apropriada a utilização do LevelDB para estrutura de dados simples como *key-values*, este é por predefinição a tecnologia utilizada. Por outro lado, o CouchDB possibilita efetuar consultas (*queries*) mais elaboradas e definir um modelo de dados mais complexo, permite guardar documentos no formato JSON.

4.5 Chaincodes

O conceito *chaincode* no HLF corresponde a um conjunto de *smart contracts*, com a mesma definição já descrita anteriormente (secção 2.2), isto é, definem em código executável regras para as organizações participantes na rede registarem transações no *world state* do *ledger*.

Este agrupamento dos *smart contracts* em *chaincodes* é sobretudo utilizado pelos utilizadores administradores da rede, facilitando a sua implantação (*deployment*) e divisão por área/contexto do negócio. Para além disso, os *chaincodes* podem classificar-se como de sistema (*system*), que visam gerir configurações do sistema. Podem ser desenvolvidos nas linguagens de programação Go[11], Java[12] e NodeJs[13].

Como já explicado (secção 4.4), o *ledger* é constituído por duas partes, sendo que os *smart contracts* conseguem aceder a ambas. Na primeira (*world state*) têm a capacidade de efetuar leituras (*get*), atualizações/modificações (*put*) e remoções (*delete*), enquanto na segunda (*blockchain*) estão limitados a leituras, pois esta parte é imutável (Hyperledger, 2020).

Todos os *chaincodes* devem possuir uma *policy* do tipo *Smart Contract Endorsement* (secção 4.2), de forma a identificar quantas ou quais organizações devem confirmar uma transação para que esta seja considerada válida. Os *chaincodes* são executados num ambiente Docker isolado de qualquer componente, a Figura 17 exemplifica a sua execução no HLF, incluindo as fases de aprovação (*endorsement*) e validação (*validation*).

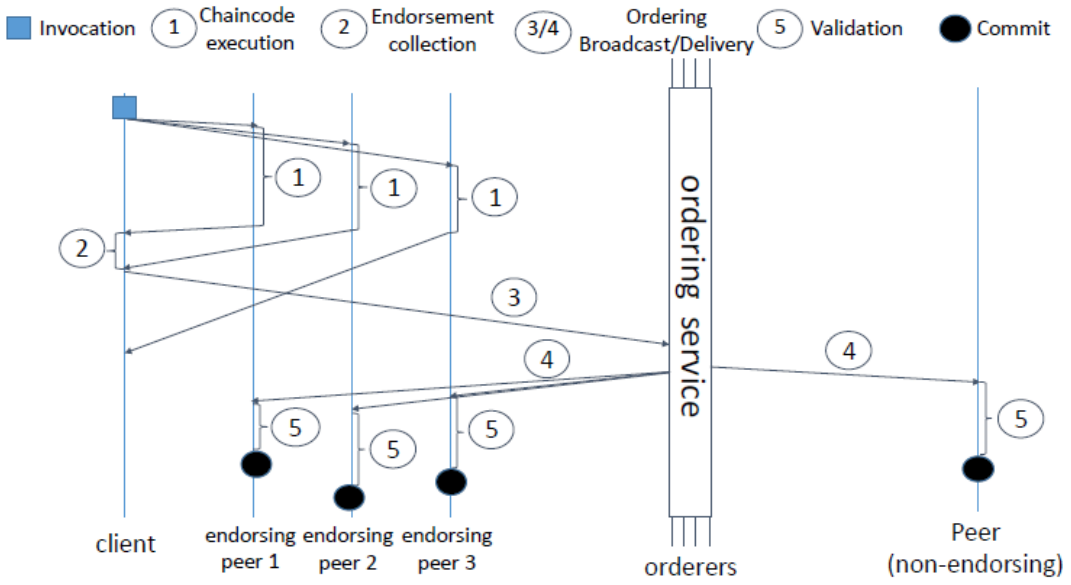


Figura 17 – Execução de um *chaincode* no HLF (Mourouzis & Tandon, 2019).

[11] Página oficial Go: 'go.dev'.
[12] Página oficial Java: 'www.java.com'.
[13] Página oficial NodeJs: 'nodejs.org'.

Com vista na Figura 17, a execução do *chaincode* (1) é invocada pelas novas transações, nesta fase os *endorsement peers* executam os *smart contracts* constituintes do *chaincode*, e em caso de sucesso assinam as transações com a finalidade de corresponder aos requisitos da *endorsement policy* definida (2). Posteriormente, através do *orderer* da rede (*ordering service*) (3 e 4) os *peers* são notificados para procederem à validação final.

Por fim, após a validação dos *endorsement peers* (5) a transação está pronta para o *commit*, e consequentemente alterar o *world state* do *ledger*, com informação atualizada. Note-se que todas as transações para além do conteúdo proposto (*proposal*), possuem um identificador único e uma resposta assinada por um conjunto de organizações da rede, ou seja, independentemente de ser ou não considerada válida é registada na *blockchain* (Mourouzis & Tandon, 2019).

Contudo, os *chaincodes* podem ser definidos ao nível de um *channel* (secção 4.3), ou seja, os seus *smart contracts* constituintes podem ser executados pelas organizações conectadas no *channel*. No entanto, qualquer definição ou alteração num *chaincode* requer aprovação das organizações descritas na *endorsement policy* (Hyperledger, 2020).

5 Análise e Desenho da Solução

Este capítulo tem como objetivo expor toda a análise e desenho efetuados para a elaboração do POC descrito neste documento (secção 1.2). Aqui, são contextualizados todos os conceitos relativos ao problema e identificadas as partes do problema onde a solução procura intervir. O capítulo inicia-se com a exploração do domínio do POC, de seguida é definido o processo de negócio e, por último, examinam-se diagramas arquiteturais da solução.

5.1 Domínio do POC

A prova de conceito (POC) é uma parte importante do desenvolvimento de um produto. É a evidência obtida de um projeto protótipo de forma a demonstrar a ideia e viabilidade de um novo produto (Malsam, 2021).

Como especificado no capítulo inicial deste documento (capítulo 0), o desenvolvimento do POC advém principalmente pelo desconhecimento e curiosidade pessoal pela área do BC, e da visão da sua aplicabilidade num contexto industrial, com o objetivo de replicar/expandir sistemas tradicionais e possibilitar novos modelos de negócio.

Vulgarmente, na indústria do aluguer de automóveis o acordo entre cliente e empresa é um ato indispensável, a empresa fornece um serviço personalizado com determinadas condições que o cliente deve cumprir, no sentido de usufruir do mesmo e não ser penalizado.

Este POC pretende apresentar um sistema que possibilita um novo processo de negócio, isto é, a ligação entre diversas empresas do mesmo ramo. Para tal, considera-se as empresas fictícias Gocar, RentaDrive e DriveNowRentals, estas têm o objetivo de se tornarem parceiras, estabelecerem acordos e partilharem os seus automóveis.

Trata-se de um processo naturalmente complexo, pela burocracia que exige e necessidade de garantia do cumprimento de todas as condições pré-acordadas entre os parceiros, de forma assegurar o mutualismo entre todos.

Por exemplo, a partilha de um automóvel entre empresas pressupõe o pagamento de um montante de aluguer, este valor pode eventualmente ser influenciado pelo estado de entrega do automóvel (e.g. apresenta danos), o tempo de utilização ultrapassou o acordado, ou a distância percorrida excedeu o limite estabelecido.

Desta forma, seria inviável a não digitalização e automatização das operações inerentes ao processo. O sistema deve contemplar que a qualquer momento pode surgir a necessidade de integrar novas empresas como participantes (com ou sem automóveis próprios), pelo que deve suportar diversos tipos de utilizadores (colaboradores das empresas) com permissões distintas.

A Figura 18 esquematiza o problema acima descrito, proposto para o desenvolvimento do POC.

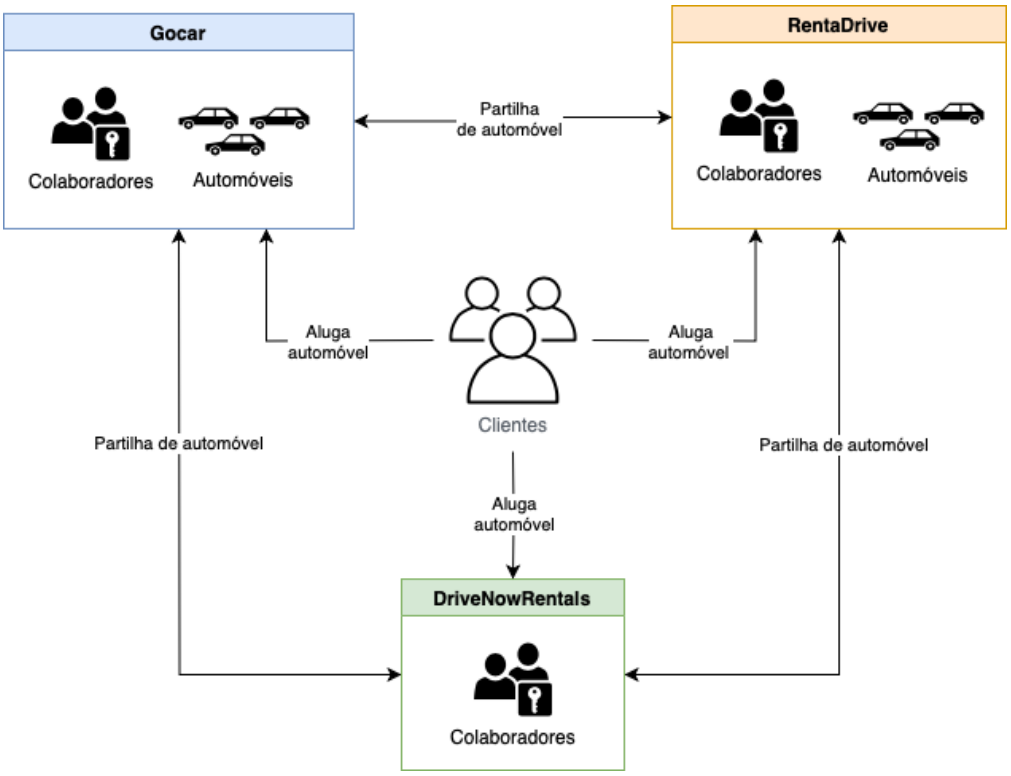


Figura 18 - Esquematização do problema proposto para o POC.

Um cliente pretende alugar à Gocar um automóvel com determinadas características, em caso de indisponibilidade, há a possibilidade desta empresa solicitar o automóvel a um parceiro, por exemplo a RentaDrive, com o objetivo de conseguir providenciar o serviço ao cliente.

Os automóveis registados no sistema são caracterizados por uma categoria, marca (fabricante), matrícula única, a disponibilidade (encontra-se elegível para aluguer), e outras características (e.g. tipo de combustível e número de lugares).

O fluxograma da Figura 19 exemplifica o processo de partilha automóvel entre a Gocar e RentaDrive.

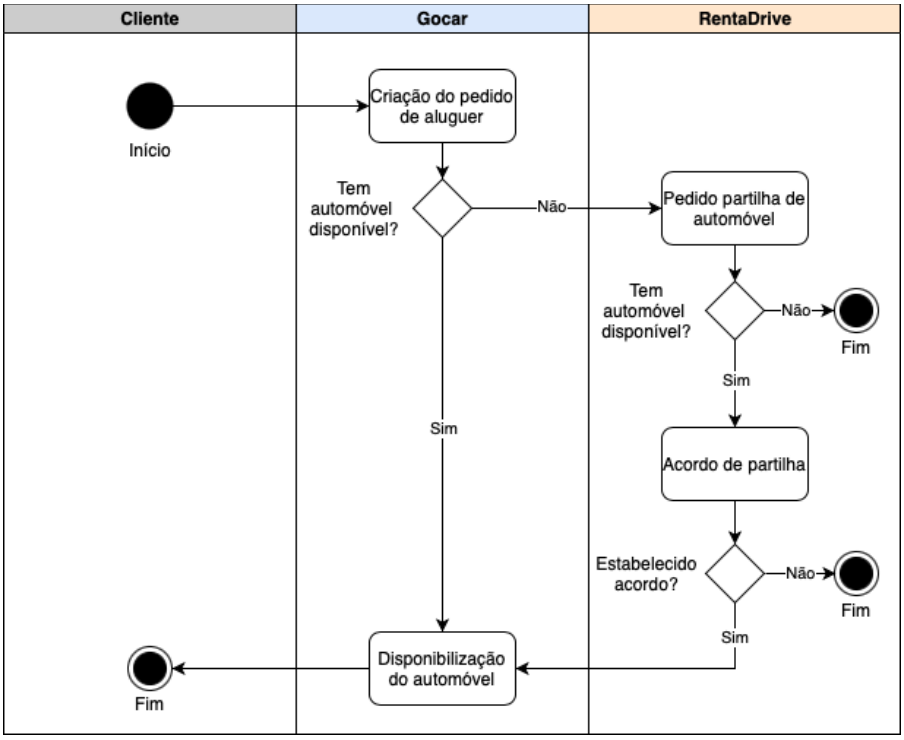


Figura 19 - Fluxograma da partilha de automóveis entre empresas.

Na partilha do automóvel entre empresas, para além da verificação da disponibilidade do mesmo, é necessário estabelecer um acordo de partilha, com a descrição das condições estabelecidas entre os parceiros. O glossário, Tabela 14, fornece a contextualização de diversos conceitos associados ao problema.

Tabela 14 - Glossário para análise do problema do POC.

Conceito	Definição
Acordo de Partilha	Contrato estabelecido entre empresas, com condições a validar na partilha dos automóveis.
Automóvel	Objeto de locomoção, que pode ser alugado por clientes e partilhado entre empresas.
Categoria (do Automóvel)	Categorização única dos automóveis (familiar, SUV, desportivo, coupé, conversível).
Cliente	Pessoa que aluga automóvel.
Colaboradores	Funcionários das empresas, que interagem com o sistema.
Empresa	Entidade que possui (ou não) automóveis, e fornece serviços de aluguer.
Matrícula (do automóvel)	Identificador único do automóvel.
Role (do Utilizador)	Função específica que o utilizador possui no sistema.
Tipo de Utilizador	Categorização dos colaboradores no sistema.

Em suma, estes são os principais aspetos atender no desenvolvimento deste POC.

5.2 Processo de negócio

Esta secção pretende fornecer uma visão objetiva do processo de negócio que o sistema visa viabilizar, descrevendo-se os principais cenários a ter em conta no desenho e implementação da solução.

Com base no problema apresentado na secção anterior (secção 5.1), o objetivo central deste novo processo de negócio é permitir a criação de acordos de partilha de automóveis entre empresas ligadas à indústria de aluguer de automóveis.

Assim, de forma atender á resolução do problema foram definidas as seguintes etapas do processo:

1. **Configuração do acordo de partilha:** etapa inicial, obrigatória para viabilizar na partilha de automóveis entre empresas;
2. **Utilização do automóvel:** após efetuado um acordo de partilha, a empresa tem a possibilidade de alugar o automóvel de outra empresa (partilhado) aos seus clientes;
3. **Devolução do automóvel:** no final do aluguer, aquando da entrega do automóvel é necessário garantir que a sua utilização não violou os termos do acordo.

Para uma compreensão mais clara do processo, o fluxograma da Figura 20 específica as etapas acima descritas. Note-se que, apesar da representação do "Cliente", o foco são as interações entre as empresas de aluguer, Gocar e RentaDrive.

O processo inicia-se pelo "Cliente" a submeter um de pedido de aluguer à Gocar. Esta verifica se possui o automóvel com as características pretendidas e, em caso negativo, poderá recorrer à conceção de um acordo de partilha com a RentaDrive. Considera-se que um acordo prévio possa já existir, não sendo repetidamente necessário a criação de um novo a cada aluguer.

Para tal, propõe um acordo de partilha com a RentaDrive, nele devem ser especificados todos os termos a considerar no empréstimo dos automóveis, período temporal da partilha e distancia máxima permitida.

Após a elaboração do acordo a RentaDrive, dependendo da disponibilidade do automóvel requerido, procede á partilha do mesmo, para que a Gocar consiga servir o "Cliente".

Por fim, posteriormente à utilização e devolução do automóvel utilizado pelo "Cliente", caso este seja partilhado, a RentaDrive (empresa proprietária) verifica se existiram irregularidades na utilização do mesmo face ao acordo de partilha, podendo originar penalizações para a Gocar.

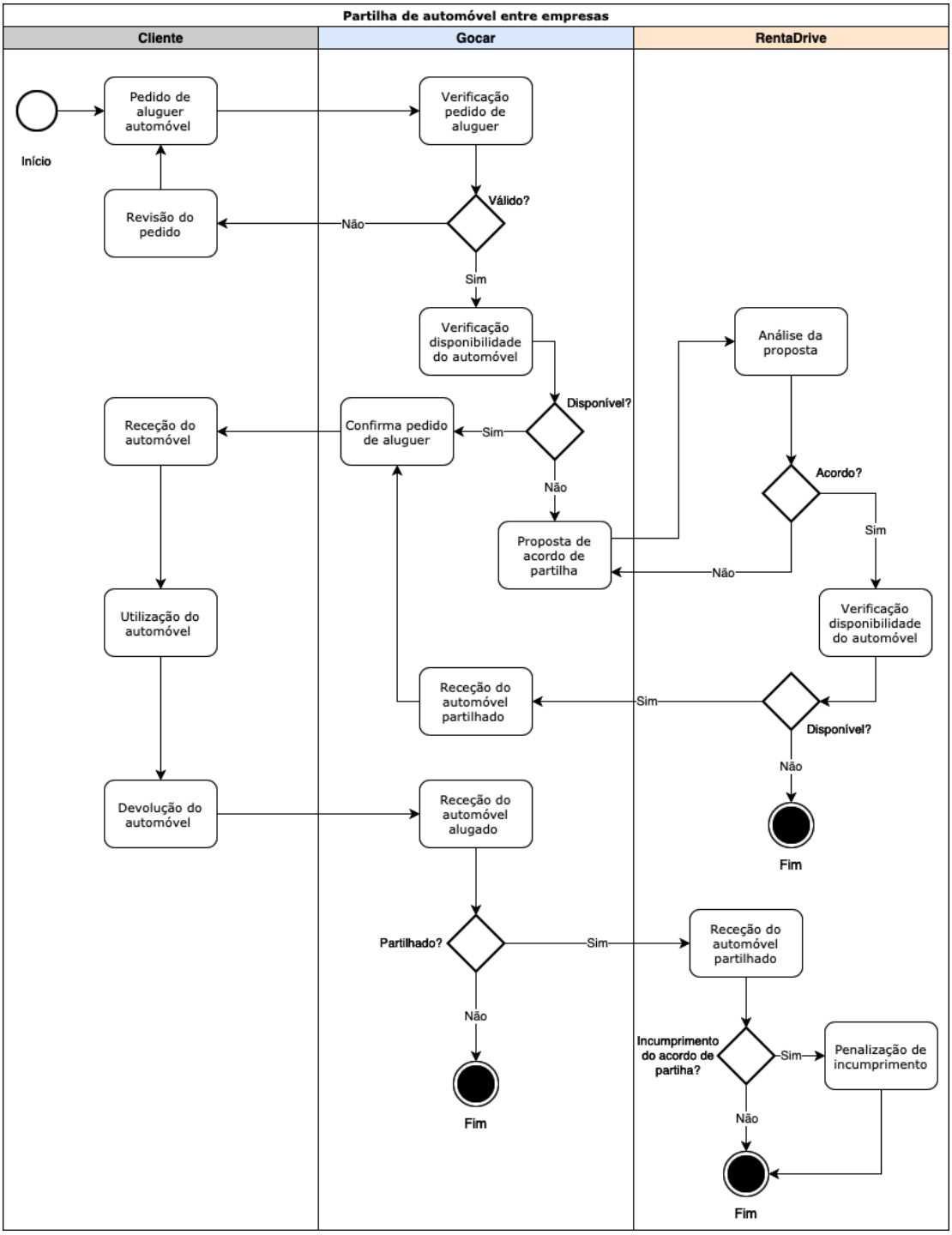


Figura 20 – Fluxograma do processo de negócio.

O sistema a desenvolver deve suportar as operações do processo supramencionado, de forma a garantir e preservar as relações empresa-empresa.

5.3 Desenho

Nesta secção são apresentadas as decisões arquiteturais relevantes tomadas no decorrer do desenvolvimento do POC. Numa primeira abordagem é determinante ter em consideração o processo de negócio já mencionado (secção 5.2), o tipo de sistema a desenvolver (secção 3.3.1), e a tecnologia HLF (capítulo 4).

Resumidamente, pretende-se criar uma BC que possibilite a integração de diversas empresas de aluguer de automóveis, com a finalidade de partilharem os automóveis mediante o estabelecimento de acordos de partilha.

A Figura 21 enquadra as diversas empresas do consórcio num sistema descentralizado.

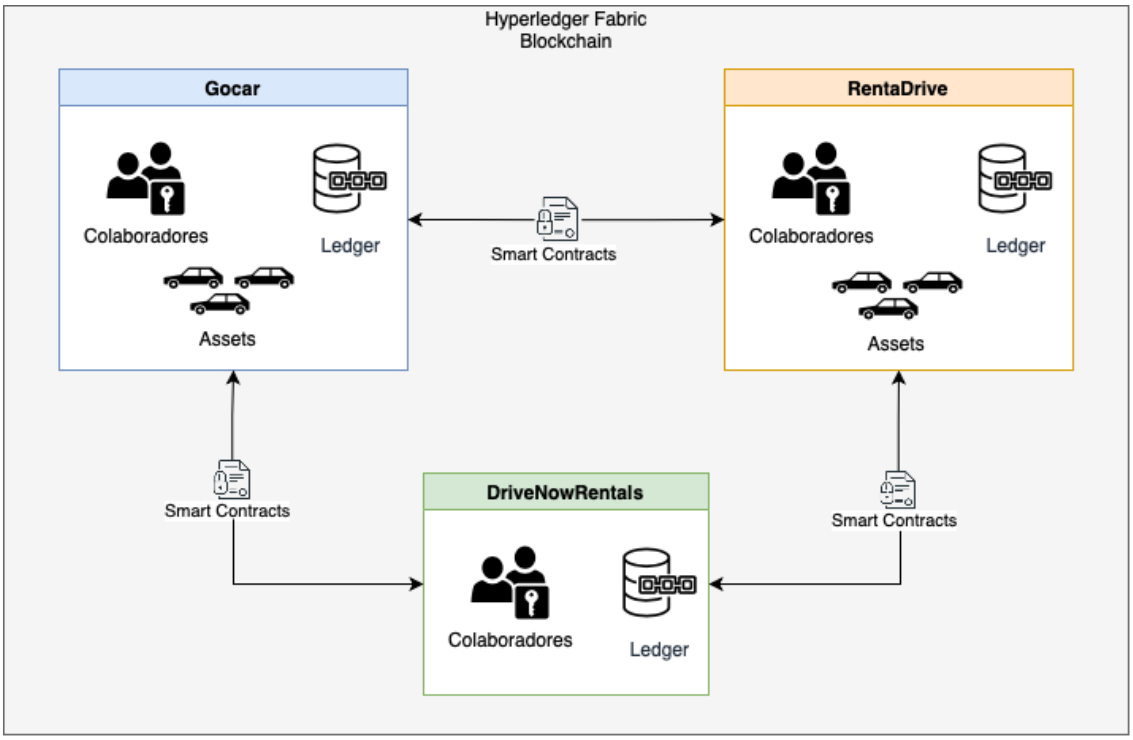


Figura 21 - Esquematização do sistema descentralizado proposto para o POC.

Identifica-se como participantes na BC as empresas de aluguer (Gocar, RentaDrive, DriveNowRentals), enquanto os automóveis, a informação dos alugueres (dos automóveis) e acordos de partilha consideram-se *assets*. Pretende-se efetuar controlo de acesso das informações que cada participante pode ler, de forma a garantir a confidencialidade de acordos de partilha.

Na elaboração do desenho descrito nesta secção, é fundamental a perceção dos conceitos do HLF descritos no capítulo 4. As empresas participantes na BC constituem um consórcio (*consortium*), denominado CarShare, visto tratar-se de um conjunto de entidades que colaboram juntamente em função de um objetivo final comum (Kenton, et al., 2022).

Por outro lado, *consortium* é também um termo nativo utilizado na configuração de uma BC no HLF. Tendo por base os conceitos da tecnologia HLF, a Figura 22 apresenta a vista lógica da BC proposta para o consórcio CarShare, onde são explicitas as interações entre os diversos componentes do sistema.

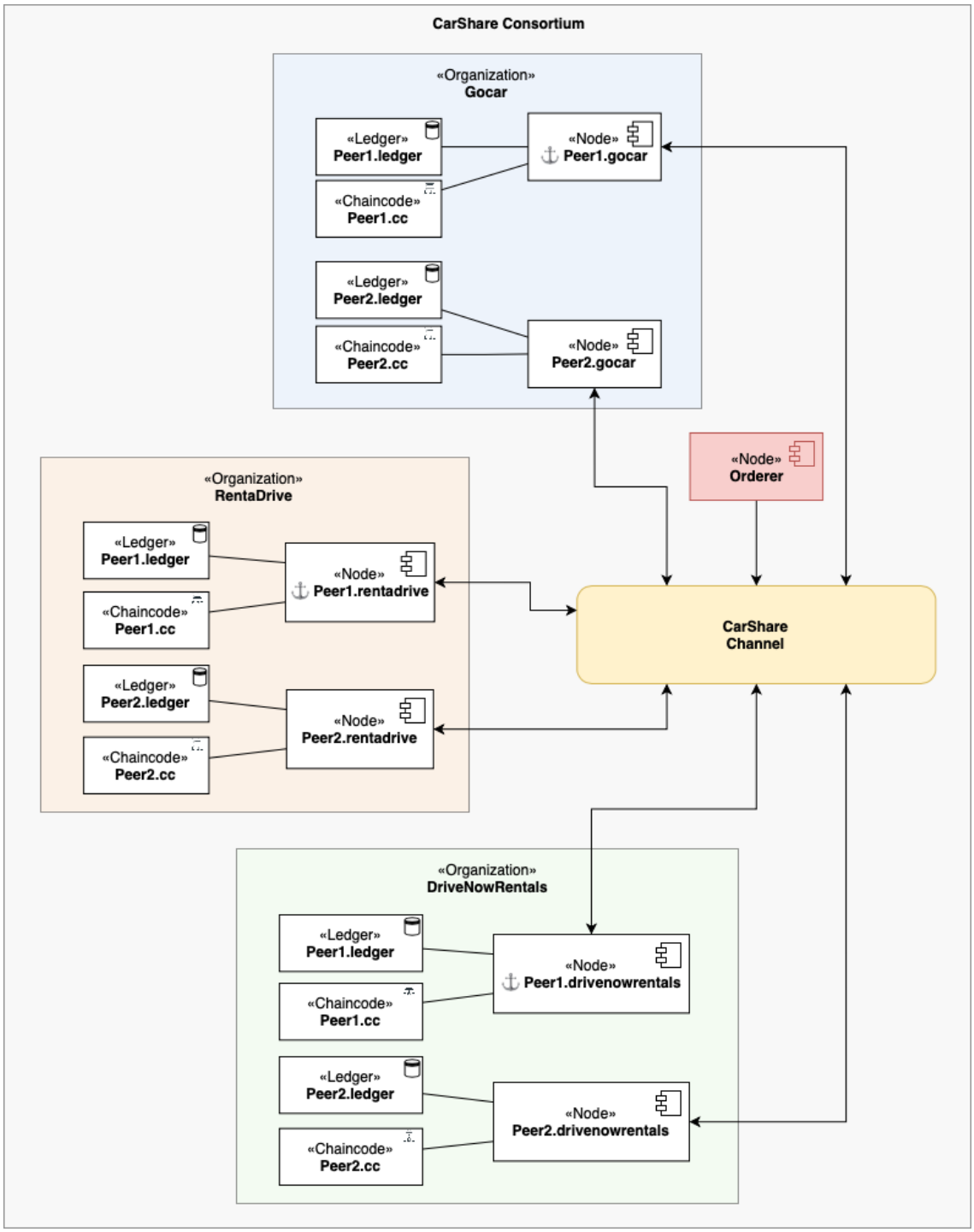


Figura 22 - Vista lógica de todos os componentes integrantes da BC do consórcio CarShare.

Com o foco na Figura 22, cada empresa é uma representada como uma "*Organization*" (ORG) que possui os *peers*, *chaincodes* e *ledgers*. O "CarShare Channel" é o *channel* (secção 4.3) definido para o consórcio constituído pela Gocar, RentaDrive e DriveNowRentals. A integração das organizações num *channel* restringe o acesso às transações efetuadas apenas aos *peers* registados no mesmo, ou seja, as informações apenas são acessíveis por componentes no contexto do *channel*.

A qualquer momento é possível configurar novos *channels* na BC, ou a adição de outras organizações aos existentes. Os *peers* "Peer1.gocar" e "Peer2.gocar" da ORG Gocar estão registados no *channel* "CarShare Channel", assim como os da RentaDrive ("Peer1.rentadrive" e "Peer2.rentadrive") e DriveNowRentals ("Peer1.drivenowrentals" e "Peer2. drivenowrentals"). O componente "Orderer" garante que as informações persistidas nos *ledgers* dos diversos *peers* das várias organizações é consistente. A execução de uma leitura através de um *chaincode* em qualquer *peer* configurado no *channel* deverá, eventualmente, retornar o mesmo resultado.

O número total de *peers* por ORG é arbitrário, embora apenas sejam apresentados dois por ORG, pode ser definido um número variável destes componentes. A configuração de diversos *anchor peer* (secção 4.3) contribui para a redundância do sistema, eliminando um possível ponto único de falha (SPOF), uma vez que estes são o primeiro ponto de contacto no momento da submissão de transações na BC.

De notar que a opção pela utilização de um único componente *orderer* (tipo Solo[14]) e dois *peer* (um do tipo *anchor*) por organização deve-se ao facto do trabalho descrito neste documento tratar-se de um POC, executável num ambiente de desenvolvimento controlado e de recursos limitados.

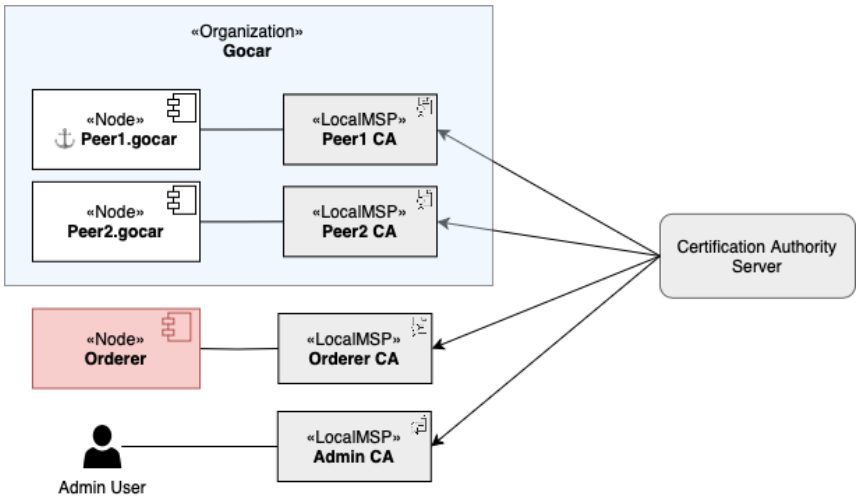


Figura 23 - Vista lógica das identidades dos componentes da BC do consórcio CarShare.

[14] Documentação oficial *Ordering Service* Hyperledger-Fabric: hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html.

A Figura 23 enquadra os CA's dos componentes da Gocar ORG ("Peer1.gocar" e "Peer2.gocar"), do *orderer* e utilizadores ("Admin User") com o componente responsável pela gestão, o *Certification Authority Server.*

Apesar de apenas apresentada a Gocar ORG na figura anterior (Figura 23), a representação é valida para qualquer componente de outra ORG.

Os Fabric Certificate Authorities (CA), como já mencionado (secção 4.1), são indispensáveis no funcionamento da BC HLF, é necessário que qualquer utilizador, componente ou aplicação disponha de certificados válidos para assinarem as transações e garantirem a sua identidade.

No decorrer dos desenvolvimentos do POC descrito neste documento serão devidamente configurados os CA's para todos os componentes constituintes do consórcio CarShare apresentados anteriormente (Figura 22).

A Figura 24 exibe o modelo de domínio elaborado para dar resposta ao processo de negócio já descrito (secção 5.2).

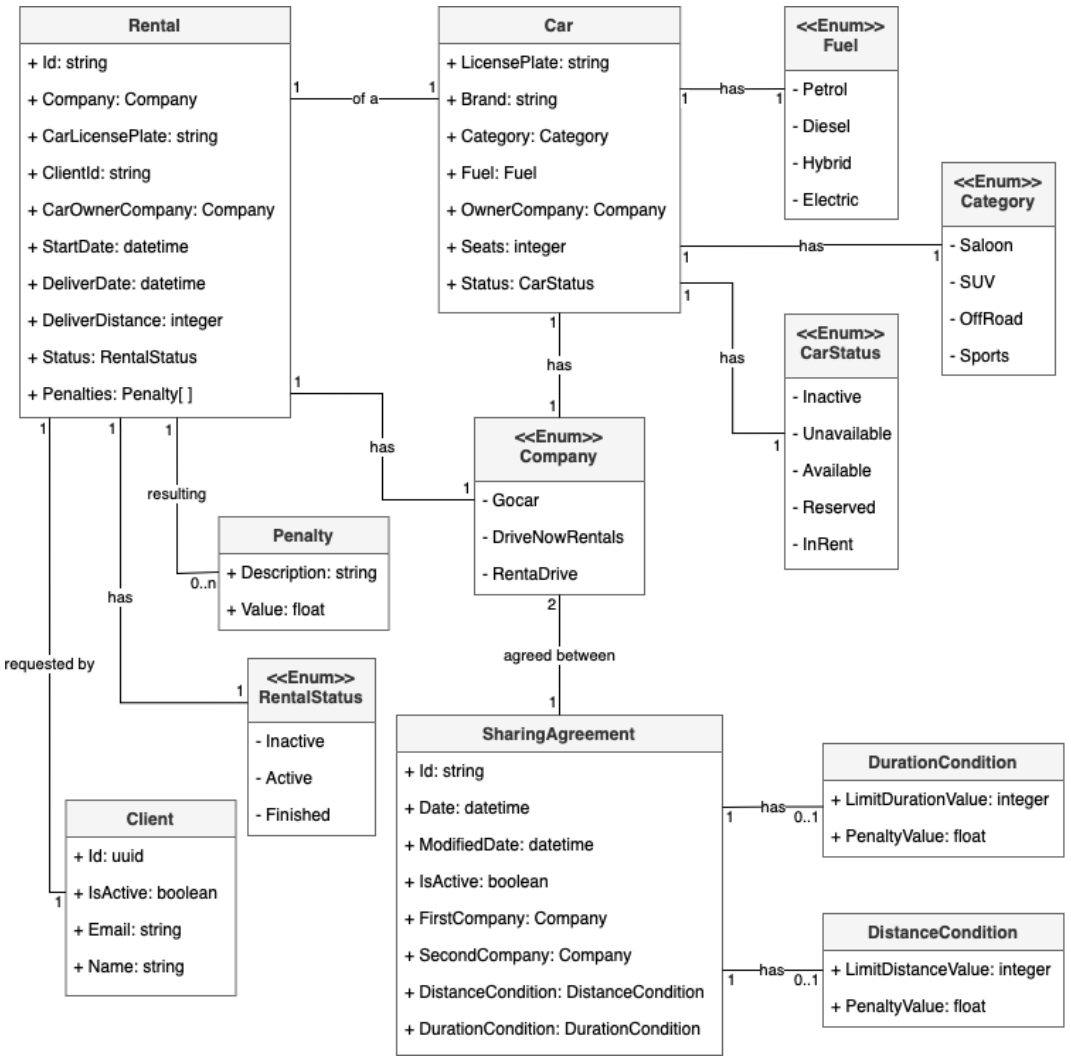


Figura 24 – Modelo de domínio definido para o processo de negócio (secção 5.2).

O foco da análise deste modelo (Figura 24) passa pela compreensão das relações entre as entidades “Car”, “Rental” e “SharingAgreement”. O acordo de partilha de automóveis (“SharingAgreement”) dá-se entre duas ORG’s (“FirstCompany” e “SecondCompany”) e define condições de utilização para o tempo e distância do automóvel cedido.

Um aluguer (“Rental”) é referente a um automóvel (“Car”) que é propriedade de uma ORG, no caso do automóvel pertencer a uma ORG distinta da qual onde é efetuado o aluguer, deve previamente existir um acordo de partilha, caso contrário o aluguer não é possível. Considerando um aluguer celebrado no âmbito de um acordo de partilha, quando aplicável, deverão ser registados (aquando do seu término) os valores de penalidade (“PenaltyValue”) previstos.

A Tabela 15 e Tabela 16 apresentam as entidades “Car” e “Rental”, respetivamente, tendo em consideração alguns os enumerados (*Enum*) associados.

Tabela 15 - Dicionário de dados da entidade "Car".

Campo	Tipo de dados	Descrição
LicensePlate	*string*	Matrícula identificadora do automóvel, única no sistema.
Brand	*string*	Marca descritiva.
Category	*Enum<Category>*	Categoria classificativa, valores possíveis do enumerado “Category”.
Fuel	*Enum<Fuel>*	Tipo de combustível, valores possíveis do enumerado “Fuel”.
OwnerCompnay	*Enum<Company>*	Empresa dona do automóvel.
Seats	*integer*	Número total de lugares.
Status	*Enum<CarStatus>*	Estado de disponibilidade, valores possíveis do enumerado “CarStatus”.

Tabela 16 - Dicionário de dados da entidade "Rental".

Campo		Tipo de dados	Descrição
Id		*string*	Identificador único do aluguer.
Company		*Enum<Company>*	Empresa prestadora do serviço.
CarLicensePlate		*string*	Matrícula identificadora do automóvel alugado.
ClientId		*string*	Identificador único do cliente.
CarOwnerCompany		*Enum<Company>*	Empresa dona do automóvel.
StartDate		*datetime*	Data de início do serviço de aluguer.
DeliveryDate		*datetime*	Data de fim/entrega do serviço de aluguer.
DeliveryDistance		*integer*	Distância percorrida pelo automóvel durante o aluguer.
Status		*Enum<RentalStatus>*	Estado/fase do aluguer, valores possíveis do enumerado “RentalStatus”.
Penalties	*Description*	*string*	Descrição da penalidade atribuída ao aluguer, após o término do serviço.
	Value	*float*	Valor da penalidade a aplicar.

Por sua vez, a Tabela 17 descreve detalhadamente a entidade "SharingAgreement" que corresponde aos acordos de partilha entre ORG's, *asset* central do processo de negócio supramencionado.

Tabela 17 - Dicionário de dados da entidade "SharingAgreement".

Campo		Tipo de dados	Descrição
Id		*string*	Identificador único do acordo de partilha.
Date		*datetime*	Data de celebração do acordo.
ModifiedDate		*datetime*	Data da última alteração no acordo.
IsActive		*boolean*	Indicador do estado do acordo (*true* – activo e *false* - inativo).
FirstCompany		*Enum<Company>*	Primeira empresa participante no acordo de partilha automóvel.
SecondCompany		*Enum<Company>*	Empresa participante no acordo de partilha (distinta da *FirstCompany*).
DistanceCondition	*LimitDurationValue*	*integer*	Limite da duração máxima dos alugueres (em dias) para os automóveis partilhados entre as duas empresas.
	PenaltyValue	*float*	Valor da penalidade a aplicar, quando ultrapassado o *LimitDurationValue*.
DurationCondition	*LimitDistanceValue*	*integer*	Limite da distância máxima percorrida durante os alugueres para os automóveis partilhados entre as duas empresas.
	PenaltyValue	*float*	Valor da penalidade a aplicar, quando ultrapassado o *LimitDistanceValue*.

O acordo de partilha de automóveis é celebrado exclusivamente entre duas ORG's ("FirstCompany" e "SecondCompany"), apesar de todas as ORG's sobescritas no CarShare *channel* estabelecerem um sincronismo das informações na BC, isto é, possuem acesso às mesmas informações, a partilha do automóvel apenas é possível na presença de "ShartingAgreement" válido, no estado ativo ("IsActive" a *true*).

A Tabela 18 descreve a entidade “Client” (clientes), estes são representados no sistema apenas como uma informação suplementar do aluguer (“Rental”), uma vez que não é a prioridade do processo de negócio.

Tabela 18 - Dicionário de dados da entidade "Client".

Campo	Tipo de dados	Descrição
Id	*uuid*	Identificador único do cliente.
IsActive	*boolean*	Indicador do estado do cliente no sistema (*true* – activo e *false* - inativo).
Email	*string*	Endereço eletrónico, único no sistema.
Name	*string*	Nome indentificador do cliente.

De forma a responder às necessidades descritas acima no processo de negócio (secção 5.2), e como objetivo central do trabalho descrito neste documento, é necessário definir um conjunto de *chaincodes*. Estes são instalados nos diversos *peers* das várias ORG’s configurados no CarShare *channel*, como indica a Figura 22, por exemplo na Gocar ORG “Peer1.cc” e “Peer2.cc”.

A versão instalada dos *chaincodes* nos vários *peers* tem obrigatoriamente de ser a mesma, para que os vários *smart contracts* constituintes não contenham divergências na lógica implementada, pois independentemente do *peer* invocado a resposta das funções de leitura (GET) e escrita (POST e PUT) deve eventualmente ser a mesma, apesar da execução dos *chaincodes* ser não determinística. Outra consideração relevante é a interação entre os vários *chaincodes* distintos, isto é, os *smart contracts* de um *chaincode* podem invocar outros *smart contracts* de outro *chaincode* para obter informação.

Assim, resultante da análise do modelo de domínio explorado previamente (Figura 24), a gestão e relação das entidades será garantida com os *chaincodes* apresentados na seguinte tabela (Tabela 19).

Dos *chaincodes* abaixo na Tabela 19, destaca-se a ligação entre o "SharingAgreementCC" e "RentalsCC", e entre o "ManageCarCC" e "RentalsCC", visto que gerem informação dependente entre si. Por exemplo, para a criação de um aluguer (responsabilidade do "RentalsCC") é necessário consultar informações dos automóveis disponíveis com recurso ao "ManageCarCC", em caso da ORG proprietária do automóvel não corresponder à ORG da criação do aluguer é necessário recorrer ao "SharingAgreementCC" para validar a existência de um acordo de partilha.

Tabela 19 – *Chaincodes* propostos para a BC do POC.

Chaincode	**Descrição**
ManageClientsCC	*Chaincode* auxiliar para a obtenção de clientes já registados, de forma a complementar a informação do aluguer.
ManageCarCC	Gestão da informação dos automóveis. Adição de novos automóveis com matrícula única no sistema. Obtenção de todos os automóveis disponíveis, assim como específicos, de forma a proceder às alterações de estado (por exemplo, "*Available*" para "*InRent*").
SharingAgreementCC	Instanciação e inativação de acordos de partilha, assim como consulta dos existentes no estado ativo, com base em duas organizações.
RentalsCC	Criação e consulta de contratos de aluguer de automóveis realizados por clientes, podendo ser ou não definidos com base num acordo de partilha. Neste caso, no decorrer da devolução do automóvel são calculadas as penalidades impostas. Necessidade de verificar e validar o estado dos automóveis e acordos de partilha.

A segregação em vários *chaincodes* possibilita atribuir responsabilidade única a cada um (*single responsability principle*), prevê-se que cada *chaincode* efetue a gestão de um único *asset*.

Desta forma, promove-se uma alta coesão e baixo acoplamento, embora este último seja negativamente impactado pela dependência de diferentes *smart contracts*. Contudo, esta interação entre vários *smart contracts* de *chaincodes* distintos contribui significativamente para não duplicação de código na sua implementação.

De salientar que o HLF não possui nativamente qualquer mecanismo para salvaguardar o correto funcionamento das aplicações dos consumidores em caso de quebra de comportamento (*breaking change*) nos *chaincodes.* Isto é, imagine-se que na atualização de versão de um *chaincode* são introduzidas alterações nas estruturas de informação, ou as funções apresentam um comportamento diferente entre versões, qualquer alteração deve ser cuidadosamente estudada para que não gere impactos inesperados, por exemplo, perda e inconsistência de informação.

Em suma, estes são os artefactos arquiteturais produzidos para o desenvolvimento do POC descrito ao longo deste documento.

6 Implementação da Solução

Neste capítulo são apresentados os detalhes de implementação da solução aos objetivos propostos inicialmente, conforme a especificação da secção do Desenho (secção 5.3). Numa primeira fase é descrita a configuração da BC e do CA *Server*, seguindo-se a secção do desenvolvimento dos *chaincodes*.

Salienta-se que para uma interpretação concisa deste capítulo, é determinante a compreensão dos conceitos do HLF descritos previamente (capítulo 4). Note-se ainda que, valores representados nas figuras são meramente exemplificativos, e toda a implementação decorreu num ambiente desenvolvimento local.

6.1 Configuração da rede

O desenvolvimento de uma BC pressupõe a configuração de um ambiente devidamente adequado, com diversas ferramentas instaladas para além do HLF, o anexo (Anexo 1) clarifica a preparação do ambiente de desenvolvimento.

Na configuração desta BC foram considerados e executados os seguintes passos:

1. **CA *Server* –** Configuração do servidor de *certification authority* e registo dos clientes de autenticação (*clients*) dos componentes do sistema (e.g. *peers*);
2. ***Ordering Service* –** Configuração do componente *orderer*;
3. ***Consortium Channel* -** Publicação do canal a ser utilizado pelo consórcio;
4. ***Peers* –** Criação dos *peers* de cada organização.

A Figura 25 apresenta a estrutura definida para a configuração dos componentes da BC.

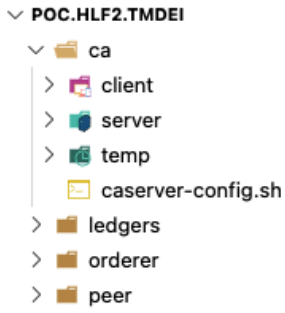


Figura 25 - Organização do diretório com os componentes HLF da BC.

O diretório raiz "POC.HLF2.TMDEI" organiza todos os componentes constituintes da BC, incluindo o servidor de CA ("ca > server") e todas os *clients* registados ("ca > client"). No sentido de automatizar alguns dos passos de configuração foram criados *scripts* com comandos do HLF *command-line interface*[15] (CLI).

Como já mencionado, no HLF todos os componentes requerem *clients* configurados no CA *server*, sendo por isso a configuração deste o primeiro passo na criação da BC. Nesta fase é também necessário definir os *roles* possíveis no sistema, considerando-se: *caserver admin* (administrador gestor do CA *server*), ORG *admins* e ORG *identities* (*peers*, *clients* e *applications*).

A configuração do CA *server* é dada pela criação do servidor e o registo dos *clients* dos utilizadores/componentes no mesmo. Para tal, o *script* "caserver-config.sh" apresentado em anexo (Anexo 2) dispõe da função "startCaServer" que utilizando o comando "fabric-ca-server start", procede à criação automática dos artefactos no diretório "server", ilustrado na Figura 26.

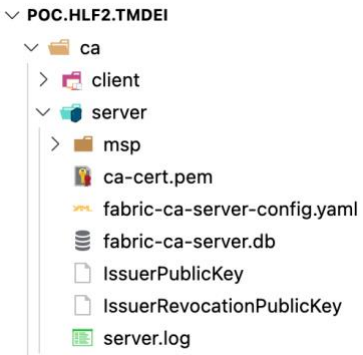


Figura 26 - Organização do diretório "server", referente ao CA *server*.

[15] Repositório oficial do HLF CLI: 'github.com/hyperledger/fabric-cli'.

Na Figura 26 destaca-se o ficheiro YAML “fabric-ca-server-config.yaml” (Anexo 3) que serve de *input* para o comando executado na função supramencionada. O ficheiro “server.log” regista toda a atividade do CA *server*, sendo por isso possível avaliar integralmente a sua execução.

Desta forma, o CA server encontra-se criado e em funcionamento, porém previamente ao registo dos utilizadores e componentes das organizações, é necessário configurar o *superuser* do servidor, para tal, executa-se a função “enrollCaAdmin” do Anexo 2, que utiliza como *input* o ficheiro YAML “fabric-ca-client-config.yaml” (Anexo 4).

Para o registo dos *clients* dos utilizadores e componentes das várias organizações (incluindo o Orderer), o procedimento é semelhante à configuração do *superuser*, são utilizadas as seguintes funções em anexo (Anexo 2), pela ordem:

- “registerOrganizationsAdmins”;
- “enrollOrganizationsAdmins”;
- “registerOrdererAdmin”;
- “enrollOrdererAdmin”;
- “registerOrganizationsUserAdmins” – opcional, utilizada para registar *clients* do tipo *user*;
- “copyCertificatesToOrganizations”.

A Figura 27 apresenta o resultado da execução das funções mencionadas acima, onde no diretório “client” constam todos os *clients* configurados, agrupados por organização quando aplicável. Note-se que todos os *clients* têm uma estrutura equivalente ao “gocar > admin”.

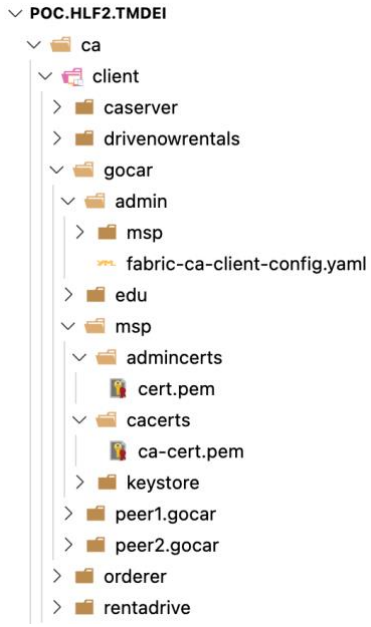
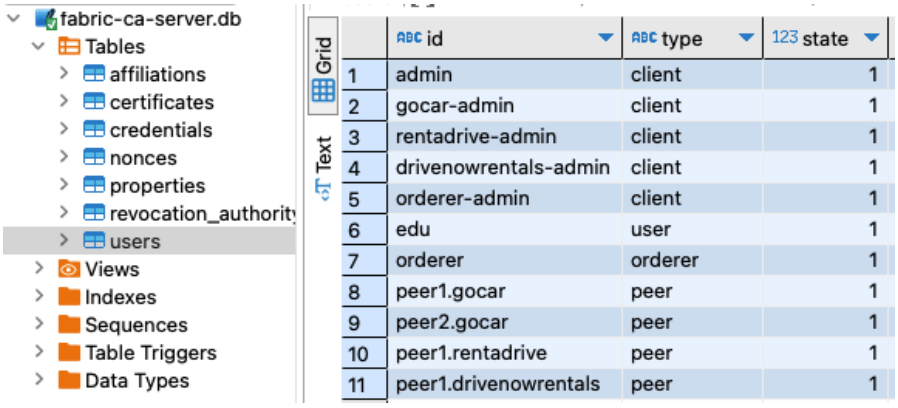


Figura 27 – Organização do diretório “ca > client” após a configuração de todos os *clients*.

No diretório “gocar > msp” (Figura 27), resultado da função “copyCertificatesToOrganizations”, encontram-se os certificados necessários para o MSP da organização, ou seja, o certificado do CA *server* (“ca-cert.pem”) e o certificado do administrador da organização (“cert.pem”).

Em suma, resultante de uma leitura (*query*) à base de dados do CA *server*, a Figura 28 exibe todos os *clients* configurados, resultantes do processo acima descrito.



	id	type	state
1	admin	client	1
2	gocar-admin	client	1
3	rentadrive-admin	client	1
4	drivenowrentals-admin	client	1
5	orderer-admin	client	1
6	edu	user	1
7	orderer	orderer	1
8	peer1.gocar	peer	1
9	peer2.gocar	peer	1
10	peer1.rentadrive	peer	1
11	peer1.drivenowrentals	peer	1

Figura 28 – *Clients* armazenados no CA *server*.

Concluído o complexo primeiro passo (1. CA *Server*), prossegue-se com a configuração do componente *orderer* (2. *Ordering Service*).

A criação deste componente dá-se no diretório “orderer” apresentado na anterior Figura 25, sendo também desenvolvido um *script*, o “orderer-config.sh”, para auxiliar na sua implementação (Anexo 5).

Dado o âmbito experimental do trabalho descrito neste documento, é possível utilizar o comando “configtxgen” do HLF CLI para gerar *crypto material*[16]. O ficheiro “configtx.yaml” em anexo (Anexo 6) divide-se em diversas secções e possui informação relativa às organizações, *ordering* service, *channel*, consórcio, *anchor peers* e *policies* gerais da rede, sendo estritamente necessário para a execução da função “createGenesisBlock” implementada no *script* referido anteriormente.

Como *output* resulta o primeiro bloco absoluto da *blockchain*, o “carshare-genesis.block”, constituído por informações definidas no ficheiro “configtx.yaml” (Anexo 6).

O *ordering service*, tal como o CA *server*, possui um ficheiro de configuração específico denominado “orderer.yaml” (Anexo 7), que em conjunto com o “carshare-genesis.block” servem de *input* para sua inicialização.

[16] O *crypto material* consiste em artefactos criptográficos gerados automaticamente pelo comando “configtxgen” do HLF CLI, isto é, auxiliar na criação de certificados com base no ficheiro “configtx.yaml” de forma facilitar os testes da rede. Note-se que o seu uso deve restringir-se apenas a ambientes de desenvolvimento.

Dada a complexidade e tamanho do bloco, a Figura 29 apresenta apenas uma fração da informação onde constam as organizações definidas para o CarShare *consortium* ("CarShareConsortium > groups").

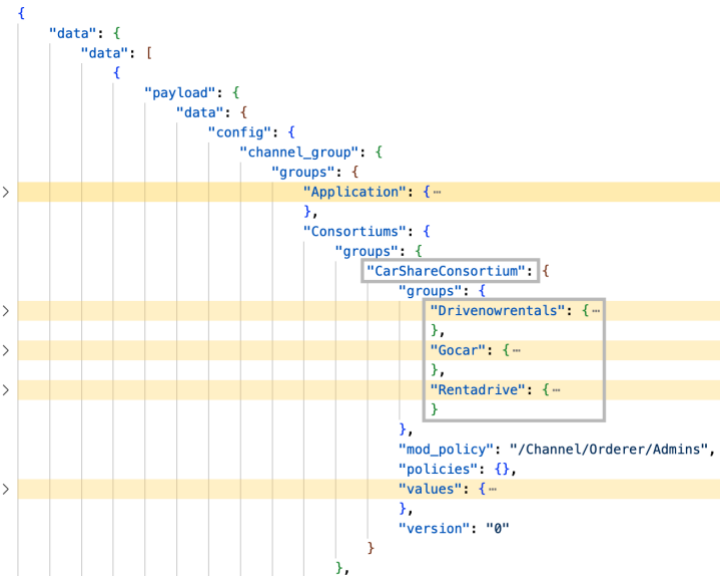


Figura 29 – Parte do resultado da desserialização do "carshare-genesis.block" para o formato JSON.

Por fim, é necessário adicionar um utilizador do tipo "orderer", que tem a responsabilidade de administrar o *ordering service*, para tal, executa-se as funções "registerOrdererUser" e "enrollOrdererUser" em anexo (Anexo 5), que registam e copiam os certificados requeridos pelo CA *server*, respetivamente.

Concluindo-se a configuração do *orderer*, é possível executar o componente através do comando "orderer", demonstrado na Figura 30.

Figura 30 - Execução do componente *orderer*.

O terceiro passo (3. *Consortium Channel*) é o mais simples da configuração descrita nesta secção. A criação de um *channel* para um consórcio de organizações, em termos técnicos, consiste apenas em criar uma transação, assiná-la com os MSP's dos administradores das organizações, e proceder à submissão da mesma. No Anexo 8 encontra-se o *script* "channel-config-sh" das funções com os comandos HLF CLI usados no procedimento.

Para a criação do CarShareChannel representado no capítulo anterior (Figura 22), a função "createChannel" executa o comando "configtxgen" que utiliza como *input* um *profile* definido no ficheiro "configtx.yaml" (Anexo 6) e origina uma transação, a "charshare-channel.tx".

Segue-se a assinatura da transação "charshare-channel.tx" pelas organizações que pretendem participar no *channel*, isto é, na função "signChannelTransactionByOrganizationAdmins" para cada organização é executado o comando "peer channel signconfigtx" que escreve na transação informação do MSP.

A Figura 31 mostra o ficheiro da transação no formato JSON após a execução da função, onde é possível verificar as assinaturas das organizações ("GocarMSP" e "DrivenowrentalsMSP") e o identificador único do *channel* ("carsharechannel").

Figura 31 - Parte do resultado da desserialização do "carshare-channel.tx" para o formato JSON.

Para finalizar a criação do *channel*, com o componente *orderer* em execução, a função "submitChannelTransactionFile" submete a transação do ficheiro "carshare-channel.tx", com recurso ao comando "peer channel create", passando o novo *channel* a estar disponível para as organizações.

Para finalizar, o quarto e último passo da configuração da rede HLF (4. *Peers*), consiste em definir os vários *peers* das organizações. Tal como outros conceitos anteriormente documentados, o componente *peer* possui também um ficheiro YAML com as respetivas propriedades de configuração, denominado "core.yaml" e equivalente ao mostrado em anexo (Anexo 10).

No *script* em anexo (Anexo 9), as funções "registerPeers" e "enrollPeers" efetuam registo dos *clients* de todos os *peers* inicialmente considerados para as várias organizações, isto é, configura-se devidamente os certificados necessários requeridos pelo CA *server*.

Atendendo à similaridade do processo, utiliza-se a criação do *anchor peer* "peer1.gocar" (secção 4.3) da organização Gocar como exemplo da configuração dos restantes *peers* na rede. A função "startPeer1GocarAndJoinChannel" atribui os valores necessários às variáveis de ambiente do HLF para a execução do *peer*, e inicia-o através do comando "peer node start".

De notar que, a única diferença entre a configuração dos tipos a*nchor* e *regular peer* consiste em adicionar previamente o endereço do *anchor peer* no "configtx.yaml" mencionado no segundo passo da configuração da rede (2. *Ordering Service*).

Ambas a funções terminam integrando os *peers* no *channel* do consórico já descrito (3. *Consortium Channel*), o comando "peer channel fetch" utiliza o bloco transacional resultante da criação do *channel* ("carsharechannel.block") e assinado por todas as organizações para descobrir o *channel* na rede, enquanto que o comando "peer channel join" integra o *peer* no *channel*. Assim, os componentes *peers* das várias organizações estão preparados para receber pedidos de aplicações que pretendam efetuar transações invocando *chaincodes*.

De modo a simplificar a inicialização dos componentes da BC configurados, é possível configurar cada um destes como um *container* Docker[17], no Anexo 11 consta parte do ficheiro *docker-compose.yml* utilizado.

Assim, conclui-se a configuração de uma rede HLF, integrando-se as organizações Gocar, DriveNowRentals e RentaDrive no consórcio CarShare. Tal como descrito, para todos os componentes e utilizadores é necessário configurar um MSP válido (1. CA *Server*) para operarem na rede, por sua vez, o componente *orderer* (2. *Ordering Service*) é determinante na garantia do *consensus* entre os participantes. O acordo para a definição de um channel como o CarShareChannel, permite às organizações controlar o acesso às transações na rede, sendo que a qualquer momento é possível a integração de novos membros. Atendendo à flexibilidade do *design* das redes HLF, o número de componentes *peer* varia conforme a necessidade das organizações.

[17] Docker é um gestor *open source* de *containers* (de aplicações). Página oficial: 'docker.com'.

6.2 Chaincodes

Esta secção é inteiramente dedicada à descrição da implementação dos *chaincodes* propostos anteriormente na secção do desenho (5.3), com o objetivo de corresponder às necessidades do processo de negócio (secção 5.2). Complementando a descrição do desenvolvimento, é também explorado o conceito *chaincode lyfecycle*, essencial na integração dos *chaincodes* na BC configurada.

Todos os *chaincodes* documentados no decorrer desta secção são desenvolvidos na linguagem de programação Golang (GO), com recurso aos *packages* da interface oficial Hyperledger Fabric Contract API[18], também denominada ContractAPI. Esta interface dispõe de um vasto conjunto de métodos necessários para a implementação dos *smart contracts* (funções) constituintes dos *chaincodes*, por exemplo, funções para o acesso de leitura e escrita ao *state* do *ledgers*.

Apresentada no final do capítulo anterior (secção 5.3), a Tabela 19 descreve sucintamente a funcionalidade de cada *chaincode*. Começando-se pelo *ManageClientsCC* que tem o objetivo de retornar informação relativa à entidade *Client* esquematizada no modelo da Figura 24 (do capítulo antecedente), este é o *chaincode* mais básico do POC, tratando-se essencialmente de um complemento e serve como o ponto de partida para a primeira interação com a linguagem Golang e interface HLF ContractAPI.

A implementação das estruturas de informação (*structs*) com os campos correspondentes a cada entidade é o primeiro passo no desenvolvimento de um *smart contract*, uma vez que todas as funções operam com base nesse modelo.

```
type Client struct {
    Id       string `json:"id"`
    IsActive bool   `json:"isActive"`
    Email    string `json:"email"`
    Name     string `json:"name"`
}
```

Código 1 – Estrutura de informação da entidade *Client*.

O Código 1 mostra a declaração da estrutura da entidade *Client*, com a definição dos tipos por atributo e a respetiva chave do JSON.

[18] Página oficial do repositório Hyperledger Fabric Contract API: github.com/hyperledger/fabric-contract-api-go.

A Tabela 20 descreve as funções implementadas no âmbito do *chaincode ManageClientsCC*.

Tabela 20 – Descrição dos *smart contracts* implementados no *chaincode ManageClientsCC*.

SmartContract	**Descrição**
InitLedger	Insere no *state* um conjunto de *clients* pré-definidos.
GetClientById	Retorna o *client* com o "Id" correspondente ao enviado por parâmetro, independentemente do estado ("IsActive" com valor *true* ou *false*).
GetAllClients	Obtém todos os *clients* persistidos no *state*, tal como o "GetClientById", sem filtragem pelo estado, e paginação.

O Código 2 corresponde à implementação do *smart contract* "*GetClientById*".

```
func (s *SmartContract) GetClientById(
    ctx contractapi.TransactionContextInterface,
    clientId string) (*Client, error)
{
    if clientId == "" {
       return nil, fmt.Errorf("provided client id is invalid: %s", clientId)
    }

    existingClientState, err := ctx.GetStub().GetState(clientId)
    if err != nil {
       return nil, fmt.Errorf("failed to read from world state")
    }

    if existingClientState == nil {
       return nil, fmt.Errorf("client with provided id doesn't exist")
    }

    var existingClient Client
    err = json.Unmarshal(existingClientState, &existingClient)
    if err != nil {
       return nil, err
    }

    return &existingClient, nil
}
```

Código 2 – *Smart contract* "GetClientById" implementado no *chaincode ManageClientsCC*.

A função "*GetClientById*" recebe como parâmetros o contexto da transação na BC e o identificador único do cliente. Previamente à pesquisa do cliente no *state* é validado o "clientId", e caso ocorra algum problema no acesso ou este não exista é retornado um erro devidamente sugestivo. Encontrado o cliente no *state*, o mesmo é convertido para a *struct* do Código 1 acima e retornado, terminando a transação com sucesso.

De notar que, o primeiro parâmetro da função (“ctx”) do tipo *TransactionContextInterface* está presente em todas as funções dos diversos *chaincodes*, pois pertence à interface HLF ContractAPI anteriormente mencionada. Sem a sua utilização, não é possível a utilização de funções que permitem interagir com o *state* da BC, como a “GetState(key string)”.

Segue-se o *chaincode ManageCarCC* que possui funções para registar novos automóveis, obter informação dos existentes, bem como, gerir o estado de disponibilidade dos mesmos para aluguer.

O Código 3 apresenta a *struct* implementada no *chaincode* para a entidade *Car*, tal como, um dos enumerados que a constituem, o *Status*.

```
type Car struct {
    Brand        string   `json:"brand"`
    Category     Category `json:"category"`
    Fuel         Fuel     `json:"fuel"`
    LicensePlate string   `json:"licensePlate"`
    OwnerCompany Company  `json:"ownerCompany"`
    Seats        int      `json:"seats"`
    Status       Status   `json:"status"`
}

type Status string
const (
    Inactive    Status = "Inactive"
    Unavailable Status = "Unavailable"
    Available   Status = "Available"
    Reserved    Status = "Reserved"
    InRent      Status = "InRent"
)
```

Código 3 – Estrutura de informação da entidade *Car* e enumerado *Status*.

A Tabela 21 apresenta as funções desenvolvidas no *chaincode ManageClientsCC*.

Tabela 21 - Descrição dos *smart contracts* implementados no *chaincode ManageCarCC*.

SmartContract	**Descrição**
InitLedger	Insere no *state* um conjunto de *cars* pré-definidos.
CreateCar	Regista um novo automóvel no *state*, com as características pretendidas.
GetCarByLicensePlate	Retorna o *car* com a matrícula (“LicensePlate”) correspondente à enviada por parâmetro.
GetAllCars	Obtém todos os *cars* persistidos no *state*, sem paginação ou filtros, independentemente do estado (“Status”) em que se encontram.
GetCarOwnerCompanyByLicensePlate	Retorna apenas a “OwnerCompany” do *car* com a matrícula (“LicensePlate”) enviada por parâmetro.
DisableCarByLicensePlate	Atualiza o “Status” do automóvel de “Available” para “Inactive”.
RentCarByLicensePlate	Atualiza o “Status” do automóvel de “Available” para “InRent”.
ReleaseCarByLicensePlate	Atualiza o “Status” do automóvel de “InRent” para “Available”.

O Código 4 abaixo mostra a função "CreateCarProposal" utilizada no *smart contract* "CreateCar", com implementação detalhada em anexo (Anexo 12 no Código 31).

```
func CreateCarProposal(
    brand string,
    category string,
    company string,
    fuel string,
    licensePlate string,
    seats int) (Car, string) {

    validationMessage := ValidateCarProposal(
       category,
       company,
       fuel,
       licensePlate,
       seats)

    if validationMessage != "" {
       return Car{}, validationMessage
    }

    return Car{
       Brand:        brand,
       Category:     GetCategory(category),
       Fuel:         GetFuel(fuel),
       LicensePlate: licensePlate,
       OwnerCompany: GetCompany(company),
       Seats:        seats,
       Status:       Available,
    }, ""
}
```

Código 4 – Função auxiliar "CreateCarProposal" implementada no *chaincode ManageCarCC*.

A função "CreateCarProposal" recebe por parâmetro os valores para os vários atributos do novo automóvel ("brand", "category", "company", "fuel", "licensePlate" e "seats"), que serão devidamente validados com a chamada à função "ValidateCarProposal" (Código 29 do Anexo 12) . Dependendo do retorno desta última, será ou não criada a nova instância do automóvel.

De forma geral, em todos os *chaincodes* desenvolvidos neste POC, os *smart contracts* responsáveis por criar registos das respetivas entidades seguem a mesma estrutura, ou seja, uma função para a criação da nova instância com o prefixo "Create" e sufixo "Proposal" no nome (Código 4), e outro com o prefixo "Validate" e mesmo sufixo (Código 29).

A Figura 32 mostra um diagrama com as transações de estados possíveis do atributo "*Status*" da entidade *Car*.

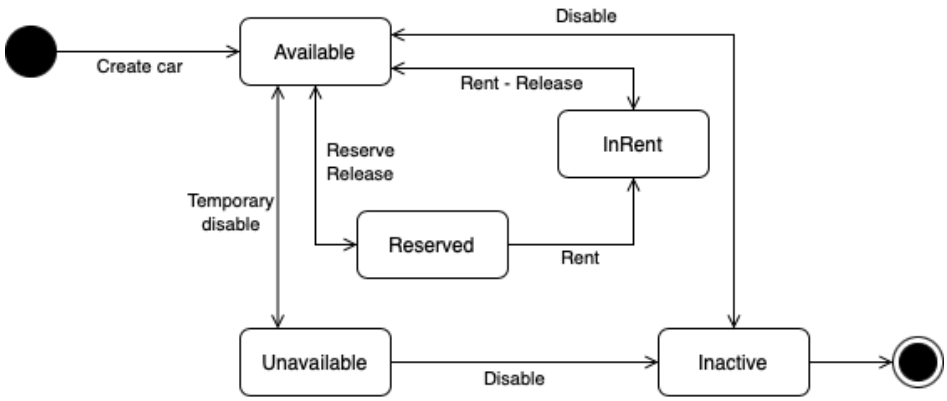


Figura 32 - Diagrama de estados referente ao "Status" da entidade *Car*.

Partindo do diagrama acima, aquando da criação de um novo automóvel o seu estado ("status") inicial é "Available", significando que está disponível para aluguer (*rental*) e que pode a partir deste estado transitar para qualquer outro. O "Inactive" é o estado final, ou seja, não é possível do mesmo efetuar qualquer alteração. Apesar de representados, neste *chaincode* não está definido nenhum caso de uso para os estados "Reserved" e "Unavailable".

O Código 5 mostra o *smart contract* implementado no *chaincode ManageCarCC* que efetua a transição do estado "Available" para "InRent".

```
func (s *SmartContract) RentCarByLicensePlate(
    ctx contractapi.TransactionContextInterface,
    licensePlate string) error {

    existingCar, _ := s.GetCarByLicensePlate(ctx, licensePlate)

    if existingCar.Status == Inactive {
       return fmt.Errorf(
           "failed release car with license plate '%s', car is 'Inactive'",
           licensePlate)
    } else if existingCar.Status != Available {
       return fmt.Errorf(
           "failed renting car with license plate '%s', car is '%s'",
           licensePlate,
           string(existingCar.Status))
    }

    return s.UpdateCarStatusByLicensePlate(ctx, existingCar, InRent)
}
```

Código 5 – *Smart contract* "RentCarByLicensePlate" implementado no chaincode *ManageCarCC*.

A função "*RentCarByLicensePlate*" efetua validações no estado atual do automóvel, para garantir que a transição de estado é possível, tal como já sugerido na Figura 32. Para além deste *smart contract*, existem outros que efetuam alterações ao estado do automóvel, nomeadamente o "*DisableCarByLicensePlate*" e o "*ReleaseCarByLicensePlate*".

Por sua vez, o *chaincode SharingAgreementsCC* dispõe de funções para a celebração de acordos de partilha de automóveis entre empresas, idealizados no processo de negócio descrito no capítulo anterior (secção 5.2). O Código 6 mostra a definição da *struct* da entidade *SharingAgreement*.

```
type SharingAgreement struct {
    Id                string            `json:"id"`
    Date              time.Time         `json:"date"`
    ModifiedDate      time.Time         `json:"modifiedDate"`
    IsActive          bool              `json:"isActive"`
    FirstCompany      Company           `json:"firstCompany"`
    SecondCompany     Company           `json:"requestingCompany"`
    DistanceCondition DistanceCondition `json:"distanceCondition"`
    DurationCondition DurationCondition `json:"durationCondition"`
}

type DistanceCondition struct {
    LimitDistanceValue int     `json:"limitDistanceValue"`
    PenaltyValue       float32 `json:"penaltyValue"`
}

type DurationCondition struct {
    LimitDurationValue int     `json:"limitDurationValue"`// Days
    PenaltyValue       float32 `json:"penaltyValue"`
}
```

Código 6 – Estrutura de informação da entidade *SharingAgreement*, da *DistanceCondition* e *DurationCondition*.

As *structs DistanceCondition* e *DurationCondition* constituintes do *SharingAgreement* apresentam especial relevância na devolução do automóvel utilizado num aluguer (*rental*) criado no âmbito de um acordo de partilha.

Na Tabela 22 estão descritas todas as funções desenvolvidas no *chaincode SharingAgreementsCC*.

Tabela 22 - Descrição dos *smart contracts* implementados no *chaincode SharingAgreementCC*.

SmartContract	**Descrição**
InitLedger	Insere no *state* um *sharing agreement* de exemplo.
CreateSharingAgreement	Regista um novo acordo de partilha entre duas empresas no *state*. Valida previamente se já existe algum acordo ativo ("IsActive" com valor *true*).
DisableSharingAgreemenById	Inativa ("IsActive" transita para *false*) um acordo existente.
GetSharingAgreementByCompanies	Retorna, caso exista, o *sharing agreement* ativo ("IsActive" a *true*) entre as duas empresas requeridas por parâmetro.

Note-se que, para a realização de novos acordos de partilha é pressuposto que as empresas já se encontrem no mesmo *channel* (secção 4.3), de forma a tornar possível o acesso aos mesmos *assets* de automóveis, clientes e alugueres.

O seguinte código (Código 7) é a implementação do *smart contract* utilizado para celebrar acordos de partilha entre duas empresas de aluguer de automóveis, denominado "*CreateSharingAgreement*".

```
func (s *SmartContract) CreateSharingAgreement(
    ctx contractapi.TransactionContextInterface,
    firstCompany string, secondCompany string,
    limitDistanceValue int, distancePenaltyValue float32,
    limitDurationValue int, durationPenaltyValue float32)
    (*SharingAgreement, error) {

    existingAgreement, _ := s.GetSharingAgreementByCompanies(
       ctx, firstCompany, secondCompany)

     if existingAgreement != nil {
       return existingAgreement,
           fmt.Errorf(
               "an active sharing agreement with %s and %s already exists",
               firstCompany, secondCompany)
    }

    sharingAgreementProposal, validationMessage :=
        CreateSharingAgreementProposal(
            firstCompany, secondCompany, limitDistanceValue,
            distancePenaltyValue, limitDurationValue,
            durationPenaltyValue)

    if validationMessage != "" {
       return nil, fmt.Errorf(validationMessage)
    }

    agreementJson, err := json.Marshal(sharingAgreementProposal)
    if err != nil {
       return nil, fmt.Errorf(err.Error())
    }

    err = ctx.GetStub().PutState(sharingAgreementProposal.Id, agreementJson)
    if err != nil {
       return nil, fmt.Errorf(
           "failed to put new sharing agreement to world state")
    }

    createdAgreement, err := s.GetSharingAgreementByCompanies(
        ctx, firstCompany, secondCompany)

    return createdAgreement, nil
}
```

Código 7 - *Smart contract* "CreateSharingAgreement" implementado no chaincode *SharingAgreementsCC*.

A primeira condição do *smart contract* (Código 7) é determinar se existe no estado ativo ("IsActive" com valor *true*) algum *SharingAgreement* para as duas empresas enviadas por parâmetro da função ("firstCompany" e "secondCompany"), utilizando a função "*GetSharingAgreementByCompanies*" (Código 32 do Anexo 13). Para além disso, são passados os valores de penalidade ("distancePenaltyValue" e "durationPenaltyValue") a aplicar sobre os alugueres de automóveis efetuados no âmbito do acordo de partilha, que ultrapassem os limites indicados ("limitDistanceValue" e "limitDurationValue").

Por último, segue-se o *RentalsCC*, o *chaincode* mais complexo elaborado neste POC. O seguinte Código 8 apresenta a definição das *structs* das entidades para o aluguer (de um automóvel) e penalidade a aplicar na devolução do mesmo, o *Rental* e *Penalty*, respetivamente.

```
type Rental struct {
    Id              string    `json:"Id"`
    Company         Company   `json:"company"`
    ClientId        string    `json:"clientId"`
    CarLicensePlate string    `json:"carLicensePlate"`
    CarOwnerCompany Company   `json:"CarOwnerCompany"`
    StartDate       time.Time `json:"startDate"`
    DeliveryDate    time.Time `json:"deliveryDate"`
    DeliveryDistance int      `json:"deliveryDistance"`
    Status          Status    `json:"status"`
    Penalties       []Penalty `json:"penalties"`
}

type Penalty struct {
    Description string  `json:"description"`
    Value       float32 `json:"value"`
}
```

Código 8 - Estrutura de informação da entidade *Rental*.

A Tabela 23 descreve sucintamente os *smart contracts* desenvolvidos no *chaincode RentalsCC*, destacando-se o "*CreateRental*" pela sua importância no processo de negócio (secção 5.2).

Tabela 23 - Descrição dos *smart contracts* implementados no *chaincode RentalsCC*.

SmartContract	**Descrição**
InitLedger	Insere no *state* um conjunto de *rentals* de exemplo.
CreateRental	Regista um novo aluguer de automóvel (*rental*) numa determinada empresa, para um automóvel (no estado "Available") e cliente existentes.
GetRentalById	Obtém um o *rental* existente para o identificador fornecido (no formato "dia-mês-ano-LicensePlate").
GetAllRentals	Obtém todos os *rentals* persistidos no *state*, sem paginação ou filtros, independentemente do estado ("Status") em que se encontram.
DeliverRentalCar	Termina o aluguer de um automóvel, alterando os estados do *rental* para "Finished" e do automóvel para "Available". Além disso, quando aplicável, aplica as penalidades previstas no *SharingAgreement* entre empresas.

Atendendo à dimensão e complexidade do *smart contract* "*CreateRental*" o diagrama de sequência da Figura 33 enquadra a sua implementação no sentido de auxiliar à sua plena compreensão. Salienta-se a dependência deste *smart contract* para outros inseridos nos diversos *chaincodes* já acima descritos, como por exemplo o "*GetClientById*" do *ManageClientsCC*.

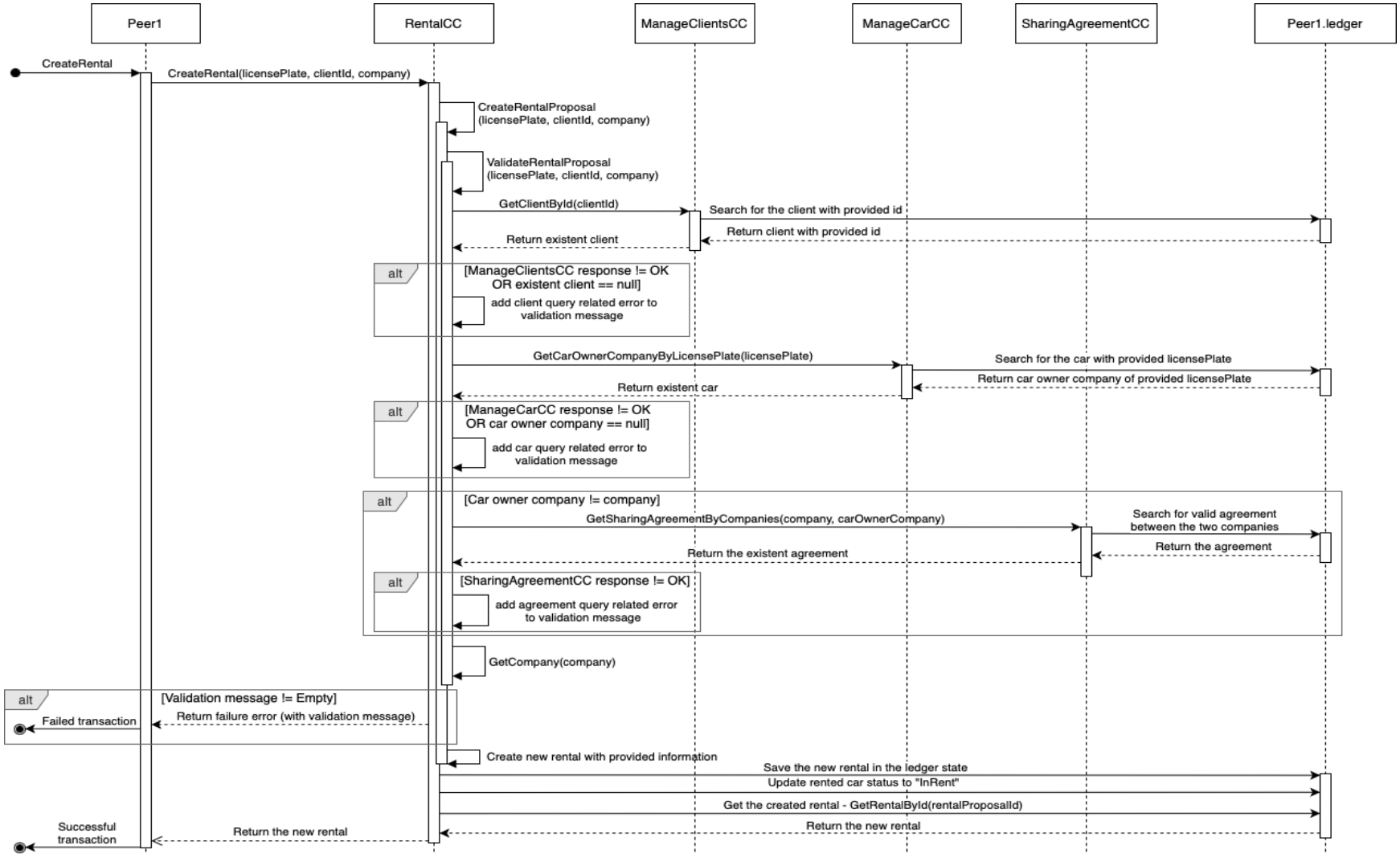


Figura 33 - Diagrama de sequência do *smart contract CreateRental* implementado no *chaincode RentalsCC*.

Com vista na *struct* do *rental* (Código 8) e diagrama de sequência da figura acima (Figura 33), entende-se por "Peer1" um componente *peer* (secção 4.3) pertencente a qualquer organização integrada no CarShareChannel, que recebe uma transação para a criação de um aluguer de automóvel. O *smart contract*, como supramencionado, apresenta dependência para *smart contracts* implementados nos vários *chaincodes*, de forma a obter e validar a informação necessária para a criação de um novo *rental*.

Para o registo de um aluguer de automóvel são necessários os dados do cliente ("clientId"), do automóvel ("licensePlate") e empresa ("company"). A validação dos dados de *input* e garantia de consistência da nova proposta (*proposal*) de *rental* é assegurada pela função auxiliar "*ValidateRentalProposal*" invocada pela função "*CreateRentalProposal*", que instancia o novo objeto.

A função "*ValidateRentalProposal*" contém toda a lógica de invocar outros *smart contracts*, de modo a consultar e validar as informações do cliente, do automóvel e quando aplicável do acordo de partilha ativo entre a empresa onde é efetuado o aluguer ("company") e a empresa proprietária do automóvel ("carOwnerCompany"). Assim, no decorrer da execução da "*ValidateRentalProposal*" são utilizados os *smart contracts* "*GetClientById*", "*GetCarOwnerCompanyByLicensePlate*" e "*GetSharingAgreementByCompanies*", dos *chaincodes ManageClientsCC*, *ManageCarCC* e *SharingAgreementCC*, respetivamente. Em caso de falha na obtenção e processamento desta informação uma mensagem de erro adequada é incrementada, e posteriormente retornada, com o objetivo de informar a aplicação que iniciou a transação do problema ocorrido.

Após a obtenção do automóvel (pelo "*GetCarOwnerCompanyByLicensePlate*") é verificado se a empresa proprietária ("company") deste é a mesma do *peer* onde a transação foi iniciada, em caso negativo, é necessária a obtenção do acordo de partilha, pelo *smart contract* "*GetSharingAgreementByCompanies*". Por fim, o estado ("Status") do automóvel transita para "InRent", o novo *rental* é persistido no *state* dos *ledgers* e de seguida retornado no resultado da transação.

De forma a complementar a descrição elaborada acima relativa ao diagrama de sequência do *smart contract* "*CreateRental*", o código da função auxiliar "*ValidateRentalProposal*" está presente em anexo (Anexo 14).

No seguimento da transação que origina um novo *rental*, é importante mencionar o ato de devolução do automóvel alugado, conforme referido anteriormente no processo de negócio (secção 5.2). Aquando do término de um aluguer, a devolução do automóvel pressupõe o apuramento das penalidades previstas no *sharing agreement* entre as empresas, quando aplicáveis. No anexo já referido (Anexo 14), encontra-se a implementação do *smart contract* "*DeliverRentalCar*" (Código 34). Este, inicialmente valida o estado do *rental*, utiliza o *smart contract* "*GetSharingAgreementByCompanies*" para obter o acordo de partilha e determinar as penalidades ("Penalties"), enquanto o *smart contract* "*ReleaseCarByLicensePlate*" altera o estado do automóvel para "Available".

Posteriormente à descrição dos *chaincodes* desenvolvidos, é relevante esclarecer de que forma se integram na BC configurada. O conceito *Fabric chaincode lifecycle* no HLF consiste nas diversas etapas, após o seu desenvolvimento, para a instalação do *chaincode* nos *peers* da BC (Hyperledger, 2023):

1. ***Packaging* –** Criação de um ficheiro compacto único do tipo *tar* com o projeto que contém o código (neste POC na linguagem GO) do *chaincode*;
2. **Instalação nos *peers* –** instalação do ficheiro *tar* criado pelo passo anterior no *peer*. Durante a execução, o *chaincode* é compilado e, em caso de sucesso, é retornado o seu identificador no formato "nome:*hash*";
3. **Aprovação pela organização –** a nova definição/versão do *chaincode* necessita ser aprovada pelas organizações integradas no *channel*, dependendo das *policies* definidas para a BC (secção 4.2);
4. ***Commit* para o *channel* –** após a aprovação devidamente efetuada no passo anterior (etapa 3), a nova definição/versão do *chaincode* está preparada para ser submetida e tornar-se executável no *channel*.

Em anexo (Anexo 15 e Código 35), encontram-se as configurações e comandos do *chaincode lifecyle* que permitem a concretização das etapas acima descritas. A instalação das várias versões dos *chaincodes* pode ser verificada pela execução do comando "peer lifecycle chaincode queryinstalled", tal como ilustrado na seguinte figura (Figura 34).

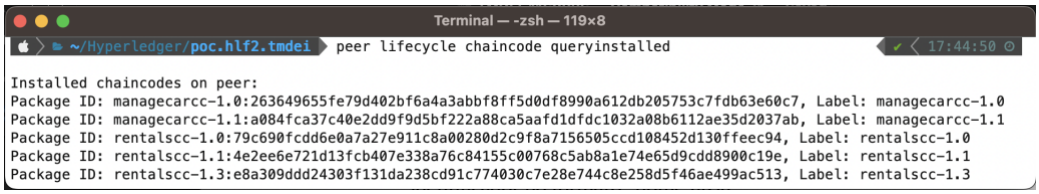


Figura 34 - Resultado da execução do comando "peer lifecycle chaincode queryinstalled" (Anexo 15).

Posteriormente à instalação do *chaincode* no *peer* é necessário a aprovação da nova versão do mesmo pelas várias organizações conforme a *signature policy* definida, pois o HLF rege-se pela governação descentralizada dos *chaincodes* (*decentralized chaincode governace*). Por exemplo, atendendo aos comandos no Anexo 15, a *policy* "OR('GocarMSP.member','DrivenowrentalsMSP.member')" significa que para o *chaincode* seja efetivamente instalado, é necessário que um membro da Gocar ou DriveNowRentals aprove a nova versão.

A Figura 35 apresenta o resultado da última fase do *chaincode lifecycle*, após a execução do comando "peer lifecycle chaincode commit" (Anexo 15).

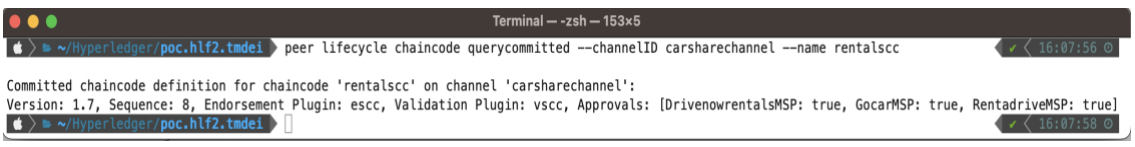


Figura 35 - Resultado da execução do comando "peer lifecycle chaincode querycomitted" (Anexo 15).

Para além de validar que o *commit* do *chaincode* ocorreu com sucesso, a figura anterior (Figura 35) permite também entender que organizações previamente o aprovaram.

Embora o HLF possibilite a invocação dos *chaincodes* através do comando "peer chaincode invoke/query", como trabalho extra ao desenvolvimento dos mesmos, foi elaborada uma *web* API REST utilizando a tecnologia NodeJs (na linguagem Typescript) e o componente Fabric Gateway[19], que disponibiliza funções para efetuar transações na BC, invocando os *chaincodes*.

O principal objetivo desta aplicação é facilitar a interação com os *chaincodes* instalados na BC, pois a alternativa seria a utilização dos comandos do *peer* ("chaincode invoke"), tornando a dinâmica pouco interativa. Na tabela abaixo (Tabela 24) é apresentada a correspondência entre os *endpoints* da API e os *smart contracts* dos *chaincodes* anteriormente descritos.

Tabela 24 – Correspondência entre os *endpoints* da REST API e os *smart contracts*.

Método HTTP	**Rota HTTP**	***Smart contract***
GET	*/api/cars*	*GetAllCars*
	/api/cars/{licensePlate}	*GetCarByLicensePlate*
	/api/cars/{licensePlate}/ownerCompany	*GetCarOwnerCompanyByLicensePlate*
	/api/clients	*GetAllClients*
	/api/clients/{id}	*GetClientById*
	/api/sharingAgreements/{companyA}/{companyB}	*GetSharingAgreementByCompanies*
	/api/rentals	*GetAllRentals*
	/api/rentals/{id}	*GetRentalById*
POST	*/api/cars/initLedger*	*InitLedger* (Tabela 21)
	/api/cars	*CreateCar*
	/api/clients/initLedger	*InitLedger* (Tabela 20)
	/api/sharingAgreements	*CreateSharingAgreement*
	/api/sharingAgreements/initLegder	*InitLedger* (Tabela 22)
	/api/rentals/initLedger	*InitLedger* (Tabela 23)
	/api/rentals	*CreateRental*
PUT	*/api/cars/{licensePlate}/rent*	*RentCarByLicensePlate*
	/api/cars/{licensePlate}/release	*ReleaseCarByLicensePlate*
	/api/rentals/{id}/deliverCar	*DeliverRentalCar*
DELETE	*/api/cars/{licensePlate}*	*DisableCarByLicensePlate*
	/api/sharingAgreements/{companyA}/{companyB}	*DisableSharingAgreementByCompanies*

Contudo, tendo por base a tabela acima (Tabela 24), o Anexo 16 apresenta a documentação Swagger[20] da *web* REST API, que permite obter uma visão mais ampla dos *endpoints* desenvolvidos, e fornece uma interface de fácil utilização para a invocação dos mesmos.

Estes foram os *chaincodes* desenvolvidos para solucionar o processo de negócio (secção 5.2) tratado neste documento, e desta forma atingir os objetivos propostos inicialmente, com um POC executável.

[19] Página official do Hyperledger Fabric-Gateway: 'hyperledger.github.io/fabric-gateway'.
[20] Página oficial Swagger: ´swagger.io'.

7 Experimentação e Avaliação

Este capítulo é reservado à apresentação da experimentação e testes efetuados ao POC descrito neste documento, com a finalidade de validar a concretização de parte dos objetivos inicialmente propostos. Para tal, são descritas as abordagens de testes consideradas, os testes unitários e de integração aos *smart contracts* desenvolvidos, e os testes de carga efetuados à BC para avaliar a sua performance.

7.1 Testes Unitários

Os testes unitários visam garantir isoladamente o correto comportamento de cada um dos *smart contracts* constituintes dos *chaincodes*, validando-se as condições descritas em código através de cenários de sucesso e insucesso.

A seguinte Tabela 25 sumariza a totalidade de testes unitários e a respetiva percentagem de cobertura de código, por *chaincode*.

Tabela 25 - Totalidade de testes unitários e percentagem de cobertura de código por *chaincode*.

Chaincode	**Número de testes**	**Cobertura do código**
ManageClientsCC	10	89.2%
ManageCarCC	33	94.3%
SharingAgreementsCC	18	95.5%
RentalsCC	27	94.8%
TOTAL	88	94.1%

Por sua vez, o Anexo 17 complementa a descrição dos testes unitários criados para garantir o correto comportamento dos *chaincodes* desenvolvidos. No total foram definidos oitenta e oito testes unitários (na mesma estrutura do Código 36 em anexo), atingindo cerca de noventa e quatro pontos percentuais de cobertura do código dos *smart contracts*. Além disso, são apresentados os relatórios de execução dos testes unitários por cada *chaincode*.

7.2 Testes de Integração

Os testes de integração são desenvolvidos na ferramenta Postman[21] e têm como objetivo invocar os *endpoints* definidos na REST API mencionada no capítulo anterior (secção 6.2), que correspondem aos *smart contracts*. A ferramenta permite adicionar condições que são executadas após efetuado o *request* no *endpoint*, com a finalidade de se verificar a resposta obtida, por exemplo, o *status code* e objetos no *response body*. Na seguinte Tabela 26, são sucintamente descritos um conjunto de cenários de teste.

Tabela 26 – Cenários definidos para os testes de integração.

	Cenário	**Pré-condições**	**Resultado esperado**
1	Criação de *Rental* sob a necessidade de *SharingAgreement*.	a. *Client* existente; b. *Car* criado e obtido com sucesso; c. *SharingAgreement* criado e obtido com sucesso.	a. *Rental* criado com sucesso.
2	Criação de *Rental* sem a necessidade de *SharingAgreement*.	a. *Client* existente; b. *Car* criado e obtido com sucesso.	a. *Rental* criado com sucesso.
3	Criação de *Rental* com um *Client* inexistente.	a. *Car* criado com sucesso;	a. *Rental* não criado; b. Resposta com erro relacionado com *Client* não encontrado.
4	Criação de *Rental* com um *Car* inexistente.	a. *Client* existente;	a. *Rental* não criado; b. Resposta com erro relacionado com *Car* não encontrado.
5	Criação de *Rental* com um *Car* que se encontra no estado 'InRent'.	a. *Client* existente; b. *Car* criado e obtido com sucesso; c. *Car Status* editado com sucesso (para 'InRent').	a. *Rental* não criado; b. Resposta com erro relacionado com estado inválido do *Car*.
6	Criação de *Rental* sob a necessidade de *SharingAgreement*, porém não existe.	a. *Client* existente; b. *Car* criado e obtido com sucesso.	a. *Rental* não criado; b. Resposta com erro relacionado com falta de *SharingAgreement*.
7	Término de um *Rental*, devolução do *Car*.	a. *Rental* existente no estado 'Active'; b. *Car* existente.	a. *Rental* transita para o estado 'Finished'; b. *Car* transita para o estado 'Available'.

No Anexo 18 encontra-se o resultado da execução dos testes de integração via Postman (Figura 44), alcançando-se um total de cinquenta e quatro testes para os sete cenários definidos. Desta forma, aliando-se os testes unitários da secção anterior (secção 7.1), assegura-se o correto funcionamento dos *chaincodes*.

[21] Página oficial Postman: 'postman.com'.

7.3 Testes de Carga

Os testes de carga apresentam como principal objetivo avaliar a capacidade de resposta da BC durante a ocorrência de períodos com um elevado número de transações simultâneas. Para a realização dos testes no POC descrito neste documento, é utilizada a ferramenta JMeter[22] executada na máquina com as especificações apresentadas na Tabela 28 do Anexo 1, sendo este desde logo um fator limitativo na performance da BC.

No sentido de minimizar os recursos consumidos pela máquina utilizada no ambiente de testes mencionada anteriormente, foram efetuadas alterações arquiteturais na BC, isto é, diminui-se para metade o número de componentes *peer* face aos considerados inicialmente na fase do desenho (Figura 22 da secção 5.3). Desta forma, é executado apenas um *container* Docker correspondente a um componente *peer* por organização, permitindo economizar os recursos de memória e processador da máquina de testes.

Para a elaboração destes testes, considerou-se *smart contracts* que acedem a *assets* diferentes, e que efetuam tanto consultas como escritas nos *ledgers*, nomeadamente o *CreateCar* (POST - /api/cars), o *GetRentalById* (GET - /api/rentals/{id}) e o *GetAllRentals* (GET - /api/rentals) (Tabela 24 da secção 6.2). A Tabela 27 apresenta os principais resultados obtidos após a execução dos testes de carga no JMeter, por *endpoint*.

Tabela 27 – Resultados obtidos da execução dos testes de carga no JMeter.

	Número de *Threads*	***Endpoint***	***Throughput (transacões por segundo)***	**Taxa de erro**	**Tempo de Execução (*segundos*)**
E1	100	POST - /api/cars	26.4	0.0%	4
		GET - /api/rentals/{id}	22.3	0.0%	
		GET - /api/rentals	26.3	0.0%	
E2	500	POST - /api/cars	28.4	0.0%	23
		GET - /api/rentals/{id}	25.1	0.0%	
		GET - /api/rentals	39.2	0.0%	
E3	1000	POST - /api/cars	21.9	1.9%	47
		GET - /api/rentals/{id}	21.3	0.0%	
		GET - /api/rentals	27.7	0.0%	
E4	3000	POST - /api/cars	20.9	12.8%	144
		GET - /api/rentals/{id}	26.2	25.2%	
		GET - /api/rentals	22.8	7.4%	

Cada execução (E) decorreu com um número incremental de *threads*, que definem a quantidade simultânea de pedidos efetuados, sendo a ordem de invocação dos vários *smart contracts* automaticamente definida pelo JMeter. De forma geral, o número de transações por segundo manteve-se constante, com exceção para o *GetRentalById* que na execução E2 apresentou um valor bastante acima da média, cerca de trinta e nove transações por segundo.

[22] Página oficial JMeter: ´jmeter.apache.org´.

Um maior número de *threads*, provoca um expectável aumento dos tempos de execução, atingindo-se dois minutos e vinte e quatro segundos para efetuar três mil pedidos a cada um dos *endpoints*, na execução E4. Consequentemente, o aumento das *threads* desta execução (E4) resulta numa considerável taxa de erro, em média quinze pontos percentuais, ou seja, do total de nove mil pedidos efetuados, cerca de mil trezentos e cinquenta falham.

Embora não apresentado na tabela anterior (Tabela 27), aumentando o número de *threads* para cinco mil, a taxa de erro ultrapassa os trinta por cento, valor que pode ser justificado pela incapacidade de a máquina do ambiente de testes dar resposta.

De notar que, dependendo do *smart contract*, existe lógica que implica consultas e escritas extra nos *ledgers*, isto é, mais processamento necessário e esforço do componente *orderer* para notificar todos os *peers* no channel e assegurar a validade da transação. Estes fatores impactam diretamente os tempos de resposta (*response time*) obtidos, que aumentam de forma significativa com o elevado do número de *threads*, principalmente nas escritas, como o caso do *CreateCar* (POST - /api/cars).

A Figura 36 mostra um gráfico obtido na execução E3 (com 1000 *threads*), com os *response time* em milissegundos por *endpoint*.

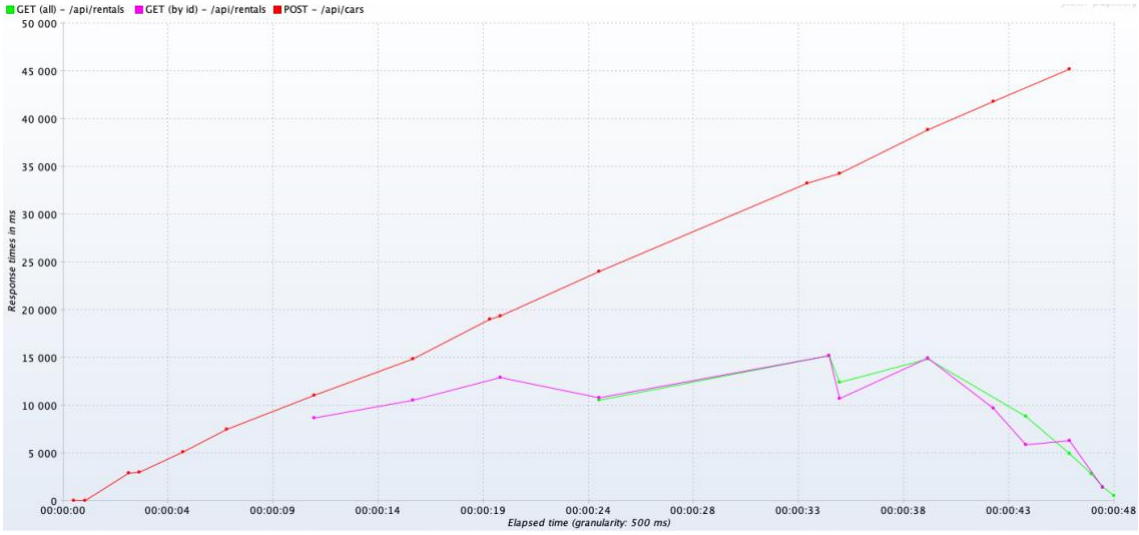


Figura 36 – Gráfico do tempo de reposta em milissegundos das transações da execução E3 (Tabela 27).

Os tempos de resposta registados pelo *endpoint* POST - /api/cars alcançam valores elevadíssimos, superiores a quarenta segundos, apesar da taxa de erro para a execução E3 ser apenas um ponto percentual. Este resultado pode ser justificado pelo acumulado de propostas de transações no componente *orderer* a aguardar a aprovação de todos os *peer*. Contudo, apesar de também elevados, os restantes *endpoints* apresentam valores constantes.

Assim, tendo por base a Análise de Valor (capítulo 3) elaborada anteriormente, o resultado obtido para o valor de transações por segundo nestes testes de carga fica bastante abaixo do valor de referência suportado pelo HLF (3500 TPS subsecção 3.3.2), obteve-se uma média de apenas setenta transações por segundo.

8 Conclusão

Este capítulo encerra o documento, apresentando um balanço geral do trabalho realizado. Desta forma, inicia-se pela discussão dos objetivos concretizados, e de seguida as limitações e possível trabalho a considerar no futuro. Para finalizar, é elaborada uma apreciação final acerca de todo o trabalho desenvolvido.

8.1 Objetivos concretizados

O trabalho iniciou-se pela definição de um problema vislumbrando a utilização da tecnologia BC e *smart contracts* (secção 1.2), com o principal objetivo final de proporcionar uma componente de investigação e desenvolvimento de um POC funcional, de caráter académico.

Dado o desconhecimento generalizado na área do BC, foram estudados os vários conceitos tecnologia (secção 2.1), assim como, casos reais de aplicabilidade da mesma (secção 2.4). Elaborou-se uma análise valor (capítulo 3) com recurso a uma técnica de análise hierárquica, proposta valor e análise funcional do problema para melhor se entender qual o tipo de sistema a desenvolver e qual projeto/tecnologia BC a utilizar.

Após cuidada documentação dos conceitos e funcionamento do HLF (capítulo 4), procedeu-se à definição do processo de negócio e posteriormente ao desenho da solução (secções 5.2 e 5.3, respetivamente), com suporte a diversos artefactos arquiteturais, etapas estas determinantes para a implementação de um POC totalmente funcional (capítulo 6), e que respondem em pleno às necessidades propostas. Para assegurar o correto funcionamento das funcionalidades incluídas no POC, criou-se um criterioso conjunto de testes de diversos tipos (capítulo 7).

Conclui-se que os objetivos do trabalho foram, de forma geral, atingidos com sucesso, cumprindo com as principais premissas de providenciar conhecimento com a componente de investigação, e o desenvolvimento de um POC utilizável na área do BC.

8.2 Limitações e Trabalho Futuro

Apesar do trabalho desenvolvido encontrar-se funcional, trata-se de um POC, e como tal a sua elaboração resulta de um vasto trabalho de investigação e aplicação dos conceitos do HLF.

Durante a exploração do HLF foi sentida a falta de suporte disponível na comunidade em geral, surgindo inúmeras dificuldades tanto na configuração da rede (secção 6.1) como na implementação dos *smart contracts* (secção 6.2), que resultaram em longos períodos de tempo para superar os problemas.

De considerar que, a tecnologia ao longo dos últimos anos tem apresentado diversas evoluções que resultam em incompatibilidades entre versões, devido à descontinuidade e adição de novas funcionalidades. Este fator dificulta o desenvolvimento pois implica, em grande parte das vezes, reconfigurar elementos da rede. Em especial, a primeira execução dos *chaincodes* implementados revelou-se desafiante, devido às distintas formas de os invocar nas várias versões do HLF, e à personalização elaborada para a rede.

Apesar do HLF suportar diversos mecanismos para promover uma maior segurança da rede, ficaram por explorar alguns deles, como por exemplo, utilização do protocolo TLS na comunicação entre os componentes, e integração de um LDAP. Relativamente às *policies* suportadas, a utilização destas deve ser mais refinada, assim como, a definição de vários tipos de utilizadores com diferentes tipos de acesso, de forma a simular uma utilização mais real do sistema.

O ambiente de execução da BC é desde logo um fator limitativo em diversos aspetos. Dada a natureza do trabalho, não foi possível obter *feedback* de utilização do sistema por utilizadores reais. No que toca a performance, tendo por base os resultados obtidos nos testes de carga (secção 7.3), seria possível paginar alguns dos resultados dos *smart contracts* de leitura, assim como, utilizar o CouchDB para possibilitar leituras mais específicas ao *state*.

Em alternativa ao SOLO, para a *stack* tecnológica do componente Orderer, seria possível considerar outras abordagens, como o Apache Kafka. O componente CA *Server* pode também ser incluído num *container* Docker.

Apesar de não explorado no âmbito deste trabalho, é possível uma integração deste POC com outros sistemas já em funcionamento nas organizações, ou seja, os *smart contracts* poderiam aceder a *web* API's e bases de dados para consultar informações existentes, sem a necessidade de uma migração total para o sistema descentralizado.

Conclui-se assim que, para além do POC desenvolvido, e atendendo à complexidade e dimensão do HLF, existem ainda diversas partes da tecnologia que podem ser exploradas, assim como diferentes abordagens no desenho do sistema.

8.3 Apreciação Final

Para finalizar, apesar de todas as dificuldades sentidas no desenvolvimento deste POC, o trabalho final resultante considera-se como bastante positivo, atendendo à concretização dos objetivos inicialmente propostos.

Todo o trabalho descrito neste documento, começando pelo estado da arte até ao detalhamento dos principais conceitos do HLF agregam seguramente conhecimento de valor para a comunidade, complementando com a extensa implementação que é um exemplar ponto de partida para futuros projetos na mesma área.

Contudo, apesar do trabalho desenvolvido atender aos objetivos, considerando as limitações e trabalho futuro já mencionados, esta primeira experiência com BC utilizando o HLF revelou-se sobretudo trabalhosa e desafiante.

Em conclusão, seria com certeza muito gratificante e recompensador a conclusão desta unidade curricular com a contribuição deste projeto de mestrado.

Referências

Hileman, D. G. & Rauchs, M., 2017. Global Blockchain Benchmarking Study. Em: Cambridge: s.n., p. 37.

Crosby, M. et al., 2016. Applied Innovation Review. *BlockChain Technology: Beyond Bitcoin*, Junho, pp. 8-9.

R3, 2022. *Blockchain/DLT 101.* [Online]
Available at: https://www.r3.com/blockchain-101/
[Acedido em 10 Fevereiro 2022].

Bitcoin, 2022. *About bitcoin.org.* [Online]
Available at: https://bitcoin.org/en/about-us
[Acedido em 10 Fevereiro 2022].

Grand View Research, 2022. *Blockchain Technology Market Size Report.* [Online]
Available at: https://www.grandviewresearch.com/industry-analysis/blockchain-technology-market
[Acedido em Fevereiro 2022].

Ethereum, 2022. *Introduction to smart contracts.* [Online]
Available at: https://ethereum.org/en/smart-contracts/
[Acedido em 10 Fevereiro 2022].

Afreen, S., 2022. *Why is Blockchain Important and Why Does it Matters.* [Online]
Available at: https://www.simplilearn.com/tutorials/blockchain-tutorial/why-is-blockchain-important
[Acedido em 12 Fevereiro 2022].

Serhii, R. & Kotik, A., 2021. *How to Implement a Blockchain in a Car Sharing Service Using the Cosmos NetworkIt was originally published on https://www.apriorit.com/.* [Online]
Available at: https://www.apriorit.com/dev-blog/733-blockchain-implement-blockchain-in-car-sharing-service-using-the-cosmos-network
[Acedido em 12 Fevereiro 2022].

Shrivas, M. K. & Yeboah, D. T., 2018. The Disruptive Blockchain: Types, Platforms and Applications. *Transformation: The Creative Potential of Interdisciplinary & Multidisciplinary Knowledge Exchange* , pp. 3-4.

Dragonchain, 2019. *What Different Types of Blockchains are There?.* [Online]
Available at: https://dragonchain.com/blog/differences-between-public-private-blockchains
[Acedido em 12 Fevereiro 2022].

Stably, 2019. *Decentralized Finance vs. Traditional Finance: What You Need To Know.* [Online]
Available at: https://medium.com/stably-blog/decentralized-finance-vs-traditional-finance-what-you-need-to-know-3b57aed7a0c2
[Acedido em 12 Fevereiro 2022].

Shrivas, M. K. & Yeboah, T., 2017. A Critical Review of Cryptocurrency Systems. Dezembro, pp. 3-4.

101 Blockchains, 2021. *Decentralized Vs. Centralized: A Detailed Comparison.* [Online]
Available at: https://101blockchains.com/decentralized-vs-centralized/
[Acedido em 11 Fevereiro 2022].

Gupta, M., 2020. *What is blockchain technology?.* [Online]
Available at: https://www.ibm.com/topics/what-is-blockchain
[Acedido em 12 Fevereiro 2022].

Afreen, S., 2022. *Why is Blockchain Important and Why Does it Matters.* [Online]
Available at: https://www.simplilearn.com/tutorials/blockchain-tutorial/blockchain-technology
[Acedido em 12 Fevereiro 2022].

Aliaga, Y. E. & Henriques, M. A., 2017. *Uma comparação de mecanismos de consenso em blockchains*, 26 e 27 Outubro, pp. 3-5.

Brede, M., 2021. [Online]
Available at: https://betterprogramming.pub/7-and-a-half-reasons-for-a-peer-to-peer-architecture-in-video-conferencing-6cb4209be3d6
[Acedido em 11 Fevereiro 2022].

Amandeep, 2020. *P2P NETWORKS.* [Online]
Available at: https://thecybercops.com/p2p-networks/
[Acedido em 11 Fevereiro 2022].

Ledger, 2019. *What are public keys and private keys?.* [Online]
Available at: https://www.ledger.com/academy/blockchain/what-are-public-keys-and-private-keys
[Acedido em 12 Fevereiro 2022].

Simplilearn, 2022. *What is Blockchain Wallet and How Does It Work?.* [Online]
Available at: https://www.simplilearn.com/tutorials/blockchain-tutorial/blockchain-wallet
[Acedido em 12 Fevereiro 2022].

Blockchain Council, 2022. *Types of Crypto Wallets Explained.* [Online]
Available at: https://www.blockchain-council.org/blockchain/types-of-crypto-wallets-explained/
[Acedido em 12 Fevereiro 2022].

Coinbase, 2022. *What is a token?.* [Online]
Available at: https://www.coinbase.com/learn/crypto-basics/what-is-a-token
[Acedido em 12 Fevereiro 2022].

Frankenfield, J. & Mansa, J., 2021. *Crypto Tokens.* [Online]
Available at: https://www.investopedia.com/terms/c/crypto-token.asp
[Acedido em 13 Fevereiro 2022].

DCX Learn, 2022. *Smart Contracts - The Complete Guide On Smart Contract Platforms.* [Online]
Available at: https://dcxlearn.com/guide/smart-contracts/
[Acedido em 12 Fevereiro 2022].

Mou, V., 2020. *Blockchain Oracles Explained.* [Online]
Available at: https://academy.binance.com/en/articles/blockchain-oracles-explained
[Acedido em 13 Fevereiro 2022].

Cryptopedia Staff, 2022. *Blockchain Oracles Explained: Decentralized Oracles in DeFi.* [Online]
Available at: https://www.gemini.com/cryptopedia/crypto-oracle-blockchain-overview
[Acedido em 15 Fevereiro 2022].

Arora, S., 2022. *What is a Smart Contract in Blockchain and How Does it Work?.* [Online]
Available at: https://www.simplilearn.com/tutorials/blockchain-tutorial/what-is-smart-contract
[Acedido em 16 Fevereiro 2022].

Afreen, S., 2022. *Why is Blockchain Important and Why Does it Matters.* [Online]
Available at: https://www.simplilearn.com/tutorials/blockchain-tutorial/why-is-blockchain-important
[Acedido em 16 Fevereiro 2022].

Essex, D., 2021. *Blockchain for businesses: The ultimate enterprise guide.* [Online]
Available at: https://www.techtarget.com/searchcio/Blockchain-for-businesses-The-ultimate-enterprise-guide
[Acedido em 14 Fevereiro 2022].

Romanovs, A. & Strebko, J., 2018. The Advantages and Disadvantages of the Blockchain Technology. Novembro, pp. 4-6.

GeeksforGeeks, 2022. *Advantages and Disadvantages of Blockchain.* [Online]
Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-blockchain/
[Acedido em 15 Fevereiro 2022].

Geroni, D., 2021. *Blockchain Scalability Problem – Why Is It Difficult To Scale Blockchain.* [Online]
Available at: https://101blockchains.com/blockchain-scalability-challenges/
[Acedido em 14 Fevereiro 2022].

Partz, H., 2022. *Daily Ethereum transactions hit a new historical high amid DeFi boom.* [Online]
Available at: https://cointelegraph.com/news/daily-ethereum-transactions-hit-a-new-historical-high-amid-defi-boom
[Acedido em 16 Fevereiro 2022].

Ethereum, 2022. *What is Ethereum?.* [Online]
Available at: https://ethereum.org/en/what-is-ethereum/
[Acedido em 16 Fevereiro 2022].

McNamara, R., 2021. *https://finance.yahoo.com/news/happened-bitcoins-transaction-volume-172321434.html.* [Online]
Available at: https://finance.yahoo.com/news/happened-bitcoins-transaction-volume-172321434.html
[Acedido em 16 Fevereiro 2022].

Crosman, P., 2017. *Banks pour $107M into blockchain consortium R3.* [Online]
Available at: https://www.americanbanker.com/news/banks-pour-107m-into-blockchain-consortium-r3
[Acedido em 16 Fevereiro 2022].

R3, 2022. *R3.* [Online]
Available at: https://www.r3.com/
[Acedido em 16 Fevereiro 2022].

Anwar, H., 2019. *Corda Blockchain: Ruler Of The Financial Enterprises.* [Online]
Available at: https://101blockchains.com/corda-blockchain
[Acedido em 16 Feveiro 2022].

Hyperledger Foundation, 2022. *What We Do.* [Online]
Available at: https://www.hyperledger.org/
[Acedido em 16 Fevereiro 2022].

Gillis, A. S., 2021. *Hyperledger.* [Online]
Available at: https://www.techtarget.com/searchcio/definition/Hyperledger
[Acedido em 16 Fevereiro 2022].

IBM, 2022. *What is Hyperledger Fabric?.* [Online]
Available at: https://www.ibm.com/topics/hyperledger
[Acedido em 16 Fevereiro 2022].

Srivastava, N., Owen, L., Bayliss, J. & Vitasek, K., 2022. *How Walmart Canada Uses Blockchain to Solve Supply-Chain Challenges.* [Online]
Available at: https://hbr.org/2022/01/how-walmart-canada-uses-blockchain-to-solve-supply-chain-challenges
[Acedido em 17 Fevereiro 2022].

Rich, N. & Holweg, M., 2000. Value Analysis. Janeiro, pp. 2-6.

Koop, C. M. & Mansa, J., 2020. *Perceived Value.* [Online]
Available at: https://www.investopedia.com/terms/p/perceived-value.asp
[Acedido em 17 Fevereiro 2022].

Andrade, J. R., 2022. Value Proposition. Em: *Value Proposition, Problem Statement e Elevator Pitch.* Porto: s.n., pp. 3-13.

Pereira, D., 2021. *The Business Model Analyst.* [Online]
Available at: https://businessmodelanalyst.com/value-proposition-canvas
[Acedido em 16 Fevereiro 2022].

Value Analysis Canada, 2021. *Function Analysis System Technique (FAST).* [Online]
Available at: https://www.valueanalysis.ca/fast.php
[Acedido em 17 Fevereiro 2022].

Araujo, S. N., 2022. *Moodle ISEP - FAST and QFD Techniques.* [Online]
[Acedido em 18 Fevereiro 2022].

Saaty, R. W., 1987. The Analytic Hierarchy Process - What it is and How it is Used. pp. 162-164.

Araujo, S. N., 2022. Método de Análise Hierarquica. Janeiro, pp. 13-30.

Lapevski, M. & Timovski, R., 2014. Analytical Hierarchical Process (AHP) Method Application in the Process of Selection and Evaluation. Novembro, pp. 373-374.

Koen, P. A. et al., 2004. Fuzzy Front End: Effective Methods, Tools, and Techniques. pp. 5-8.

Walmart, 2022. *History.* [Online]
Available at: https://corporate.walmart.com/about/history

Mukherjee, S., 2022. *How Walmart uses blockchain to manage its supply chain.* [Online]
Available at: https://analyticsindiamag.com/how-walmart-uses-blockchain-to-manage-its-supply-chain

Apgar, D., 2022. *Walmart Canada Is Using Blockchain to Ease Supply Chain Burdens.* [Online]
Available at: https://www.paymentsjournal.com/walmart-canada-is-using-blockchain-to-ease-supply-chain-burdens/

Galea-Pace, S., 2020. *The key benefits to Walmart Canada's new blockchain solution.* [Online]
Available at: https://supplychaindigital.com/digital-supply-chain/key-benefits-walmart-canadas-new-blockchain-solution

Derecha, V., 2023. *Distributed Ledger Frameworks Comparison: Corda vs Hyperledger Fabric.* [Online]
Available at: https://labs.eleks.com/2021/04/distributed-ledger-frameworks-comparison-corda-vs-hyperledger-fabric.html

Abrol, A., 2022. *Hyperledger Vs Corda Vs Ethereum: A Detailed Comparison.* [Online]
Available at: https://www.blockchain-council.org/blockchain/hyperledger-vs-corda-vs-ethereum/

Malsam, W., 2021. *What Is Proof of Concept (POC)? Definition, Steps & Best Practices.* [Online]
Available at: https://www.projectmanager.com/blog/proof-of-concept-definition

Corda, 2018. *Transactions Per Second (TPS).* [Online]
Available at: https://corda.net/blog/transactions-per-second-tps/

Agarwal, U. et al., 2022. Blockchain Technology for Secure Supply Chain Management: A Comprehensive Review. Em: s.l.:IEEE Access, p. 10.

Gorenflo, C., Lee, S. & S. Keshav, L. G., 2019. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. Em: s.l.:s.n., pp. 7-8.

Kenton, W., Estevez, E. & Reeves, M., 2022. *Consortium: Definition, Examples, Vs. Joint Venture.* [Online]
Available at: https://www.investopedia.com/terms/c/consortium.asp

Point, C., 2023. *What is an IT Security Policy?.* [Online]
Available at: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-it-security/it-security-policy/

Hyperledger, 2022. [Online]
Available at: https://uploads-ssl.webflow.com/6243075ff83d08a79dc7b307/624edb8ad39af33432e9c472_what_is_etherisc_1.0_en.pdf

Etherisc, 2022. *What is Etherisc.* [Online]
Available at: https://uploads-ssl.webflow.com/6243075ff83d08a79dc7b307/624edb8ad39af33432e9c472_what_is_etherisc_1.0_en.pdf

Ethereum, 2023. *Decentralized identity.* [Online]
Available at: https://ethereum.org/en/decentralized-identity/#what-makes-decentralized-identifiers-possible

R3, 2023. *Consensus.* [Online]
Available at: https://docs.r3.com/en/platform/corda/4.8/open-source/key-concepts-consensus.html

Hyperledger, 2023. *Open, Proven, Enterprise-grade DLT.* [Online]
Available at: https://www.hyperledger.org/wp-content/uploads/2020/03/hyperledger_fabric_whitepaper.pdf

Hyperledger, 2020. *Identity Certificate Authorities.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/identity/identity.html#root-cas-intermediate-cas-and-chains-of-trust

Hyperledger, 2020. *Membership Service Provider.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/membership/membership.html

Hyperledger, 2020. *Policies.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/policies/policies.html

Packt, 2023. *The Ledger.* [Online]
Available at: https://www.packtpub.com/book/data/9781839218750/6/ch06lvl1sec57/the-ledger
[Acedido em Abril 2023].

Natarajan, H., Krause, S. & Gradstein, H., 2017. *Distributed Ledger Technology (DLT) and Blockchain.* [Online]
Available at: https://openknowledge.worldbank.org/server/api/core/bitstreams/5166f335-35db-57d7-9c7e-110f7d018f79/content
[Acedido em Abril 2023].

Hyperledger, 2020. *Ledger.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/ledger/ledger.html#what-is-a-ledger
[Acedido em Abril 2023].

Mourouzis, T. & Tandon, J., 2019. Introduction to Decentralization and Smart Contracts. Em: s.l.:s.n., pp. 6-7.

Hyperledger, 2020. *Smart Contracts and Chaincode - Ledger.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/smartcontract/smartcontract.html#ledger
[Acedido em Abril 2023].

Hyperledger, 2020. *Smart Contracts and Chaincode - Channels.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/smartcontract/smartcontract.html#channels
[Acedido em Abril 2023].

Hyperledger, 2020. *Peers.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/peers/peers.html#peers
[Acedido em Abril 2023].

Hyperledger, 2020. *Applications and Peers.* [Online]
Available at: https://hyperledger-

fabric.readthedocs.io/fa/latest/peers/peers.html#applications-and-peers
[Acedido em Abril 2023].

Abbas, A., 2023. *What is gossip protocol?.* [Online]
Available at: https://www.educative.io/answers/what-is-gossip-protocol
[Acedido em Maio 2023].

Sakhuja, R., 2018. *Hyperledger Fabric Peer Roles.* [Online]
Available at: http://www.bcmentors.com/knowledge-base/hyperledger-fabric-peers-roles/
[Acedido em Maio 2023].

Oracle, 2023. *Manage Channels.* [Online]
Available at: https://docs.oracle.com/en/cloud/paas/blockchain-cloud/usingoci/manage-channels.html#GUID-96A2465D-FB30-4B3F-88AD-74C8FB7376E0
[Acedido em Maio 2023].

Hyperledger, 2020. *Peers - Peers and Organizations.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/peers/peers.html#peers-and-organizations
[Acedido em Maio 2023].

Hyperledger, 2020. *Orderers and the transaction flow.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/orderer/ordering_service.html#orderers-and-the-transaction-flow
[Acedido em Maio 2023].

Hyperledger, 2020. *Peers and Orderers.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/fa/latest/peers/peers.html#peers-and-orderers
[Acedido em Maio 2023].

Hyperledger, 2023. *Fabric Chaincode Lifecycle.* [Online]
Available at: https://hyperledger-fabric.readthedocs.io/en/latest/chaincode_lifecycle.html
[Acedido em 20 Julho 2023].

Anexo 1 Configuração do Ambiente de Desenvolvimento HLF

Este anexo apresenta o de ambiente utilizado para o desenvolvimento do POC utilizando a tecnologia HLF. A Tabela 28 apresenta as principais especificações da máquina utilizada no desenvolvimento do POC descrito neste documento.

Tabela 28 - Especificações da máquina de desenvolvimento.

Sistema Operativo	macOs Ventura 13.4
Memória RAM	16Gb DDR4
Processador	Intel i7 2,6GHz 6-Core

Note-se que as especificações apresentadas possuem impacto direto nos resultados obtidos na execução, nomeadamente na experimentação e avaliação (capítulo 7).

Para o desenvolvimento do POC, destaca-se a instalação das seguintes ferramentas:

- HLF[23] - tecnologia BC (capítulo 4);
- Docker[24] - gestor *open source* de *containers* (de aplicações);
- Go[25] – linguagem de programação *open source*, criada pela Google.

A Figura 37 apresenta as versões instalas das ferramentas acima mencionadas.

```
~/H/poc.hlf2.tmdei docker --version && go version && orderer version && peer version
Docker version 20.10.21, build baeda1f
go version go1.20.4 darwin/amd64
orderer:
 Version: v2.5.1
 Commit SHA: 2aa7b87
 Go version: go1.20.3
 OS/Arch: darwin/amd64

peer:
 Version: v2.5.1
 Commit SHA: 2aa7b87
 Go version: go1.20.3
 OS/Arch: darwin/amd64
 Chaincode:
  Base Docker Label: org.hyperledger.fabric
  Docker Namespace: hyperledger
```

Figura 37 - Versões instaladas do Docker, Go e HLF na máquina de desenvolvimento.

Em suma, estas são os requisitos necessários para concretizar a configuração de uma BC utilizando a tecnologia HLF.

[23] Página oficial HLF: 'hyperledger.org'.
[24] Página oficial Docker: 'docker.com'.
[25] Página oficial Go: 'go.dev'.

Anexo 2 Configuração do Servidor CA

O *script* "caserver-config.sh" abaixo (Código 9, Código 10 e Código 11), possui as funções necessárias para a criação do CA *server* e registo dos *clients* dos componentes (e.g. *peers*, utilizadores). De notar que na implementação são utilizados diversos comandos do HLF CLI, onde inicialmente são definidos alguns caminhos absolutos para a localização dos artefactos no ambiente de desenvolvimento. Apesar de apenas apresentados artefactos relativos à Gocar as configurações são aplicadas a todas as organizações.

```
# Environment variables
CA_SERVER=./poc.hlf2.tmdei/ca/server
CASERVER_CA_CLIENT=./poc.hlf2.tmdei/ca/client/caserver
GOCAR_CA_CLIENT=./poc.hlf2.tmdei/ca/client/gocar
ORDERER_CA_CLIENT=./poc.hlf2.tmdei/ca/client/orderer

# Start/Enable CA Server (creates all needed files)
function startCaServer {
    killall fabric-ca-server 2> /dev/null
    export FABRIC_CA_SERVER_HOME=$CA_SERVER
    fabric-ca-server start 2> $CA_SERVER/server.log &
    echo "CA Server Started!"
    echo "Check execution logs (server.log)"
}

function enrollCaAdmin {
    mkdir -p $CASERVER_CA_CLIENT/admin
    export FABRIC_CA_CLIENT_HOME=$CASERVER_CA_CLIENT/admin
    fabric-ca-client enroll -u http://admin:pw@localhost:7054
}

# Register/Add default admins to all Organizations (Gocar example)
function registerOrganizationsAdmins {
    export FABRIC_CA_CLIENT_HOME=$CASERVER_CA_CLIENT/admin
    ATTRIBUTES='"hf.Registrar.Roles=peer,user,client",
                "hf.AffiliationMgr=true","hf.Revoker=true"'

    mkdir -p $GOCAR_CA_CLIENT/admin
    fabric-ca-client register --id.type client --id.name gocar-admin
        --id.secret pw --id.affiliation gocar --id.attrs $ATTRIBUTES
}
```

Código 9 – Parte um do código do script "caserver-config.sh".

```
# Enroll default admins to all Organizations (Gocar example)
# Add certificate of CA Client (client/admin/msp/admincerts)
function enrollOrganizationsAdmins {
    export FABRIC_CA_CLIENT_HOME=$GOCAR_CA_CLIENT/admin
    fabric-ca-client enroll -u http://gocar-admin:pw@localhost:7054
    mkdir -p $GOCAR_CA_CLIENT/admin/msp/admincerts
    cp $CASERVER_CA_CLIENT/admin/msp/signcerts/*
        $GOCAR_CA_CLIENT/admin/msp/admincerts
}

# Register/Add default Orderer admin
function registerOrdererAdmin {
    export FABRIC_CA_CLIENT_HOME=$CASERVER_CA_CLIENT/admin
    ATTRIBUTES='"hf.Registrar.Roles=orderer"'
    mkdir -p $ORDERER_CA_CLIENT/admin
    fabric-ca-client register --id.type client --id.name orderer-admin
        --id.secret pw  --id.affiliation orderer --id.attrs $ATTRIBUTES
}

# Enroll default admin to Orderer
# Add certificate of CA Client (client/admin/msp/admincerts)
function enrollOrdererAdmin {
    export FABRIC_CA_CLIENT_HOME=$ORDERER_CA_CLIENT/admin
    fabric-ca-client enroll -u http://orderer-admin:pw@localhost:7054
    mkdir -p $ORDERER_CA_CLIENT/admin/msp/admincerts
    cp $CASERVER_CA_CLIENT/admin/msp/signcerts/*
        $ORDERER_CA_CLIENT/admin/msp/admincerts
}

# Register Organizations Users as Admins
# Add User "Edu" as Admin of gocar Organization, with area1 Affiliation
# Copy CA Admin certificates (/signcerts) to other User Admins (/admincerts)
function registerOrganizationsUserAdmins {
    export FABRIC_CA_CLIENT_HOME=$GOCAR_CA_CLIENT/admin
    fabric-ca-client register --id.type user --id.name edu
        --id.secret pw --id.affiliation gocar.area1

    export FABRIC_CA_CLIENT_HOME=$GOCAR_CA_CLIENT/edu
    fabric-ca-client enroll -u http://edu:pw@localhost:7054
    mkdir -p $GOCAR_CA_CLIENT/edu/msp/admincerts
    cp -r $GOCAR_CA_CLIENT/admin/msp/signcerts/*
        $GOCAR_CA_CLIENT/edu/msp/admincerts
}
```

Código 10 - Parte 2 do código do script "caserver-config.sh".

```
# Copy CA Server root certificate (ca-cert.pem) to Organizations MPS's (e.g.
/client/gocar/msp/cacerts)
# Copy Organization Admin certificate (e.g. gocar/admin/msp/signcerts/cert.pem)
to Organizations MPS's (e.g. /client/gocar/msp/admincerts)
function copyCertificatesToOrganizations {
    mkdir -p $GOCAR_CA_CLIENT/msp/admincerts
    mkdir -p $GOCAR_CA_CLIENT/msp/keystore
    mkdir -p $GOCAR_CA_CLIENT/msp/cacerts
    cp $CA_SERVER/ca-cert.pem $GOCAR_CA_CLIENT/msp/cacerts
    cp $GOCAR_CA_CLIENT/admin/msp/signcerts/* $GOCAR_CA_CLIENT/msp/admincerts
}
```

Código 11 - Parte três do código do script "caserver-config.sh".

Por sua vez, a Tabela 29 descreve sucintamente as funções desenvolvidas no *script* apresentado acima.

Tabela 29 - Descrição da funcionalidade das funções do *script* "caserver-config.sh".

Função	Descrição
startCaServer	Função inicial, que inicia a execução do CA *server* na porta configurada no ficheiro "fabric-ca-server-config.yaml", através do comando "fabric-ca-server start".
enrollCaAdmin	*Enroll* do administrador do CA *server*, através do comando "fabric-ca-client enroll". Este é o "superuser" do sistema.
registerOrganizationsAdmins	Registo de administradores das diversas organizações (Gocar, RentaDrive e DriveNowRentals), com atribuição de todos o *roles*. Utilização do comando "fabric-ca-client register", com os parâmetros necessários (*type*, *name*, *secret*).
enrollOrganizationsAdmins	Inscrição (*enroll*) dos administradores registados na função *registerOrganizationsAdmins*. Utilização do comando "fabric-ca-client enroll", especificando os *names* e *secrets* dos administradores.
registerOrdererAdmin	Registo do nó Orderer, com o comando "fabric-ca-client register".
enrollOrdererAdmin	*Enroll* do nó Orderer, com o comando "fabric-ca-client enroll".
registerOrganizationsUserAdmins	Exemplo de criação e registo de um utilizador, com privilégios de administrador (utilizador "edu"), com os comandos "fabric-ca-client register" e "fabric-ca-client enroll".
copyCertificatesToOrganizations	Função auxiliar para a criação do MSP e cópia do certificado ("ca-cert.pem") do CA *server* e certificados de administrador para os respetivos diretórios dos administradores registados para cada uma das organizações (Gocar, RentaDrive e DriveNowRentals).
showIdentityList	Apresentação das entidades registadas nas funções anteriores (comando "fabric-ca-client identity list").

Anexo 3 Ficheiro de Configuração do Servidor CA

O ficheiro "fabric-ca-server-config.yaml" no Código 12 e Código 13 mostram os valores atribuídos na configuração do CA *server*.

```
port: 7054
debug: false
crlsizelimit: 512000
ca:
  name: carshare-ca
  keyfile:
  certfile: ca-cert.pem
  chainfile:
affiliations:
  gocar:
    - area1
    - area2
    - area3
  orderer:
    - area1
  drivenowrentals:
    - area2
  rentadrive:
    - area2
registry:
  maxenrollments: -1
  identities:
    - name: admin
      pass: pw
      type: client
      affiliation: ""
      attrs:
        hf.Registrar.Roles: "*"
        hf.Registrar.DelegateRoles: "*"
        hf.Revoker: true
        hf.IntermediateCA: true
        hf.GenCRL: true
        hf.Registrar.Attributes: "*"
        hf.AffiliationMgr: true
```

Código 12 – Parte um do ficheiro de configuração "fabric-ca-server-config.yaml" utilizado para o CA Server.

```
csr:
  cn: Carshare-CA
  names:
    - C: US
      ST: "New York"
      L: Newark
      O: Carshare
      OU: B2B
  hosts:
    - localhost
  ca:
    expiry: 131400h
signing:
  default:
    usage:
      - digital signature
    expiry: 8760h
  profiles:
    ca:
      usage:
        - cert sign
        - crl sign
      expiry: 43800h
      caconstraint:
        isca: true
        maxpathlen: 0
    tls:
      usage:
        - signing
        - key encipherment
        - server auth
        - client auth
        - key agreement
      expiry: 8760h
tls:
  enabled: false
crl:
  expiry: 24h
ldap:
  enabled: false
db:
  type: sqlite3
  datasource: fabric-ca-server.db
bccsp:
  default: SW
  sw:
```

```
        hash: SHA2
        security: 256
```

Código 13 - Parte dois ficheiro de configuração "fabric-ca-server-config.yaml" utilizado para o CA Server.

A Tabela 30 descreve algumas das secções e respetivas propriedades do ficheiro de configuração apresentado acima (Código 12 e Código 13).

Tabela 30 - Descrição das secções e propriedades do ficheiro "fabric-ca-server-config.yaml".

Secção	Propriedade	Descrição
port	-	Define a porta (*port*) onde o CA *server* é executado.
ca	-	Informação geral relacionada com o CA *server*.
	name	Nome do servidor, “carshare-ca”.
	certificate	Nome do certificado de permissão gerado, “ca-cert.pem”.
affiliations	-	Designação de possíveis departamentos das organizações. Funcionam de forma hierárquica.
registry	-	Configuração da informação das *identities* iniciais.
registry	*identities*	Especificação do administrador do CA *server* (“superuser”), com os campos *name*, palavra-passe (*pass*), *type* e atributos (*attrs*).
csr	-	Controlo da criação do CA raiz, C*ertificate Signing Request*.
	cn	Nome do CA, “Carshare-CA”.
	ca	Prazo de expiração do certificado (em horas).
tls	*enabled*	Configuração de servidor de *Transport Layer Security*.
ldap	*enabled*	Integração do *Lightweight Directopry Access Protocol* (LDAP).
db	*type*	Indicação do tipo de base de dados a utilizar pelo servidor (“sqlite3”, “postgres” ou “mysql”).
	datasource	Especificação da localização dos dados, “fabric-ca-server.db”.
bccsp	-	Serviço de encriptação utilizado pelo, *Blockchain Crypto Service Provider*. Caracterização do serviço configurado, neste caso, SHA256.

Anexo 4 Ficheiro de Configuração dos Clients no Servidor CA

O ficheiro "fabric-ca-client-config.yaml" no Código 14, exemplifica os valores atribuídos na configuração de um *client* do CA *server*. Note-se que algumas das secções e/ou propriedades do ficheiro poderão ser sobrepostas por parâmetros utilizados nos comandos do HLF CLI.

```
url: http://localhost:7054
mspdir: msp
csr:
  cn: admin
  keyrequest:
    algo: ecdsa
    size: 256
  serialnumber:
id:
  name: gocar-admin
  type: client
  affiliation:
  maxenrollments: 0
  attributes:
    - name: hf.Registrar.Roles
      value: peer,user,client
bccsp:
    default: SW
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            keystore: msp/keystore
```

Código 14 - Exemplo de ficheiro “fabric-ca-client-config.yaml”, usado na configuração de *client*.

A Tabela 31 descreve as secções mais relevantes do ficheiro de configuração mostrado anteriormente (Código 14).

Tabela 31 - Descrição das secções e propriedades do ficheiro "fabric-ca-client-config.yaml".

Secção	Propriedade	Descrição
url	-	Endereço de execução do CA *server*.
mspdir	-	Caminho relativo para o diretório do *Membership Service Provider*.
csr	-	Controlo da criação do CA raiz, C*ertificate Signing Request*.
	cn	Nome do CA, “admin”.
id	-	Dados caracetristicos do *client*.
	name	Nome do *client*, “gocar-admin”.
	type	Indicação do tipo de *client* (“peer”, “user”, “app”, outros).
bccsp	-	Serviço de encriptação utilizado pelo, *Blockchain Crypto Service Provider*. Caracterização do serviço configurado, neste caso, SHA256.

Anexo 5 Configuração do Orderer

O Código 15 e Código 16 apresenta o *script* "orderer-config.sh", utilizado para a configuração do componente o*rderer*. Note-se que este tem precedência do *script* do Anexo 2, e como anteriormente já referido, no seu início são definidos caminhos para a localização dos artefactos no ambiente de desenvolvimento e na implementação das funções são utilizados comandos do HLF CLI (e.g. "configtxgen").

```
ORDERER_CA_CLIENT=./poc.hlf2.tmdei/ca/client/orderer
ORDERER_PATH=./poc.hlf2.tmdei/orderer
CA_SERVER=./poc.hlf2.tmdei/ca/server

# Copy Orderer Admin certificate (e.g. orderer/admin/msp/signcerts/cert.pem)
# to Orderer MPS's (e.g. /client/orderer/msp/admincerts)
function copyOrdererAdminCertificatesToMSP {
    mkdir -p $ORDERER_CA_CLIENT/msp/admincerts
    mkdir -p $ORDERER_CA_CLIENT/msp/keystore
    mkdir -p $ORDERER_CA_CLIENT/msp/cacerts

    cp $CA_SERVER/ca-cert.pem $ORDERER_CA_CLIENT/msp/cacerts
    cp $ORDERER_CA_CLIENT/admin/msp/signcerts/*
        $ORDERER_CA_CLIENT/msp/admincerts
}

# Create Genesis Block of CarShare Consorsium - Use of command configtxgen
function createGenesisBlock {
    export FABRIC_LOGGING_SPEC=DEBUG
    export FABRIC_CFG_PATH=$ORDERER_PATH

    configtxgen -profile CarShareOrdererGenesis
        -outputBlock ./carshare-genesis.block -channelID ordererchannel
}

# Register/Add "orderer" user
function registerOrdererUser {
    export FABRIC_CA_CLIENT_HOME=$ORDERER_CA_CLIENT/admin
    mkdir -p $ORDERER_CA_CLIENT/orderer

    fabric-ca-client register --id.type orderer --id.name orderer
        --id.secret pw --id.affiliation orderer
}
```

Código 15 – Parte um do código do *script* "orderer-config.sh".

```
# Enroll default "orderer" user - Add certificate of CA Client
(client/admin/msp/admincerts)
# Copy Orderer Admin certificates (/signcerts) to other orderer Admin
(/admincerts)
function enrollOrdererUser {
    export FABRIC_CA_CLIENT_HOME=$ORDERER_CA_CLIENT/orderer
    fabric-ca-client enroll -u http://orderer:pw@localhost:7054
    mkdir -p $ORDERER_CA_CLIENT/orderer/msp/admincerts

    cp $CA_SERVER/ca-cert.pem $ORDERER_CA_CLIENT/msp/cacerts

    cp $ORDERER_CA_CLIENT/admin/msp/signcerts/*
        $ORDERER_CA_CLIENT/orderer/msp/admincerts
}
```

Código 16 - Parte dois do código do script "orderer-config.sh".

A tabela abaixo (Tabela 32) descreve as funções desenvolvidas no *script* do componente *orderer*.

Tabela 32 - Descrição da funcionalidade das funções do script "orderer-config.sh".

Função	Descrição
copyOrdererAdminCertificatesToMSP	Função auxiliar para a criação do MSP e respetiva cópia do certificado ("ca-cert.pem") do *orderer*.
createGenesisBlock	Criação do bloco *genesis* da *blockchain*, necessário para inicializar a rede.
registerOrdererUser	Registo do utilizador *orderer*, para administrar o component, com o comando "fabric-ca-client register".
enrollOrdererUser	*Enroll* do utilizador *orderer*, com o comando "fabric-ca-client enroll".

Anexo 6 Ficheiro de configuração do Channel

O ficheiro "configtx.yaml" utilizado para gerar o *crypto material* é extenso e constituído por várias secções, assim sendo dividiu-se as várias secções por diversas fragmentos de código, com o objetivo de facilitar a sua interpretação. O Código 17 apresenta a secção "Organizations", de notar que as configurações do *orderer* ("&Orderer"), DriveNowRentals ("&DriveNowRentals") e RentaDrive ("&RentaDrive"), foram ocultadas devido à semelhança com a Gocar ("&Gocar").

```
Capabilities:
  Application: &ApplicationCapabilities
    V2_0: true
  Orderer: &OrdererCapabilities
    V2_0: true
  Channel: &ChannelCapabilities
    V2_0: true
Organizations:
  - &Orderer
  - &Drivenowrentals
  - &Rentadrive
  - &Gocar
    Name: Gocar
    ID: GocarMSP
    MSPDir: ./poc.hlf2.tmdei/ca/client/gocar/msp
    Policies: &GocarPolicies
      Readers:
          Type: Signature
          Rule: "OR('GocarMSP.member')"
   Writers:
          Type: Signature
          Rule: "OR('GocarMSP.member')"
      Admins:
          Type: Signature
          Rule: "OR('GocarMSP.admin','OrdererMSP.admin')"
      Endorsement:
          Type: Signature
          Rule: "OR('GocarMSP.member')"
    AnchorPeers:
      - Host: peer1.gocar
        Port: 7051
```

Código 17 - Secção "Organizations" do ficheiro "configtx.yaml".

Por sua vez, a Tabela 33 descreve brevemente as propriedades da secção “Organizations” e, tal como referido anteriormente, apenas para uma das organizações (“Gocar”).

Tabela 33 - Descrição das propriedades da secção “Organizations” ficheiro "configtx.yaml".

Propriedade	Descrição
Name	Nome da organização.
ID	Identificador único do MSP da organização.
MSPDir	Caminho para o diretório do MSP da organização.
Policies	Diversas *policies* definidas por operação, indicando o tipo e condição em que se aplicam.
AnchorPeers	*Peers* do tipo *anchor* definidos para a organização, indicando o seu endereço de execução *Host* e *Port*. Poderão ser indicados múltiplos *anchor peer*.

A Código 18 mostra a configuração do *orderer* no ficheiro "configtx.yaml".

```
Orderer: &OrdererDefaults
  OrdererType: solo
  Addresses:
    - localhost:7050
  Policies:
    Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
    Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
    Admins:
        Type: ImplicitMeta
        Rule: "ANY Admins"
    BlockValidation:
        Type: ImplicitMeta
        Rule: "ANY Writers"
  BatchTimeout: 2s
  BatchSize:
    MaxMessageCount: 10
    AbsoluteMaxBytes: 98 MB
    PreferredMaxBytes: 512 KB
  Capabilities:
    <<: *OrdererCapabilities
```

Código 18 - Secção "Orderer" do ficheiro "configtx.yaml".

A seguinte tabela (Tabela 34) mostra o significado das várias propriedades da secção do código acima (Código 18).

Tabela 34 - Descrição das propriedades da secção “Orderer” ficheiro "configtx.yaml".

Propriedade	Descrição
OrdererType	Tecnologia utilizada para na implementação do *orderer*. Descritos anteriormente na secção 4.3 Peer e Channel.
Adresses	Endereço (com a porta) onde o *orderer* é executado.
Policies	Diversas *policies* definidas por operação, indicando o tipo e condição em que se aplicam.
BatchTimeout	Intervalo de tempo definido para a criação de *batches*.
BatchSize	Define o tamanho dos *batch* de blocos de transações.
Capabilities	Ativação de funcionalidades do *ordering service* na *blockchain*, declaradas no início do ficheiro (Código 18) (dependente da versão).

Posteriormente à secção “Orderer”, a Código 19 apresenta a secção “Application”.

```
Application: &ApplicationDefaults
  ACLs: &ACLsDefault
    lscc/ChaincodeExists: /Channel/Application/Readers
    lscc/GetDeploymentSpec: /Channel/Application/Readers
    lscc/GetChaincodeData: /Channel/Application/Readers
    lscc/GetInstantiatedChaincodes: /Channel/Application/Readers
    qscc/GetChainInfo: /Channel/Application/Readers
    qscc/GetBlockByNumber: /Channel/Application/Readers
    qscc/GetBlockByHash: /Channel/Application/Readers
    qscc/GetTransactionByID: /Channel/Application/Readers
    qscc/GetBlockByTxID: /Channel/Application/Readers
    cscc/GetConfigBlock: /Channel/Application/Readers
    cscc/GetConfigTree: /Channel/Application/Readers
    cscc/SimulateConfigTreeUpdate: /Channel/Application/Readers
    peer/Propose: /Channel/Application/Writers
    peer/ChaincodeToChaincode: /Channel/Application/Readers
    event/Block: /Channel/Application/Readers
    event/FilteredBlock: /Channel/Application/Readers
    _lifecycle/CheckCommitReadiness: /Channel/Application/Writers
    _lifecycle/CommitChaincodeDefinition: /Channel/Application/Writers
    _lifecycle/QueryChaincodeDefinition: /Channel/Application/Readers
    _lifecycle/QueryChaincodeDefinitions: /Channel/Application/Readers
  Policies: &ApplicationDefaultPolicies
    Endorsement:
        Type: ImplicitMeta
        Rule: "ANY Endorsement"
    Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
    Writers:
        Type: ImplicitMeta
```

```
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
        LifecycleEndorsement:
            Type: ImplicitMeta
            Rule: "ANY Endorsement"
    Capabilities:
        <<: *ApplicationCapabilities
```

Código 19 - Secção "Application" do ficheiro "configtx.yaml".

Referente à secção "Application", a Tabela 35 sintetiza de forma breve as propriedades definidas.

Tabela 35 - Descrição das propriedades da secção "Application" ficheiro "configtx.yaml".

Propriedade	Descrição
ACLs	Configuração das *access control lists*, mencionadas na secção 4.2 Policies.
Policies	Diversas *policies* definidas por operação, indicando o tipo e condição em que se aplicam.
Capabilities	Ativação de funcionalidades das aplicações da *blockchain*, declaradas no início do ficheiro (Código 19) (Dependem da versão do HLF).

A Código 20 mostra a secção "Channel", contextualizada na secção 4.3 Peer e Channel.

```
Channel: &ChannelDefaults
    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "ANY Admins"
    Capabilities:
        <<: *ChannelCapabilities
```

Código 20 - Secção "Channel" do ficheiro "configtx.yaml".

A tabela abaixo (Tabela 36), descreve as propriedades da secção "Channel".

Tabela 36 - Descrição das propriedades da secção "Channel" ficheiro "configtx.yaml".

Propriedade	Descrição
Policies	Diversas *policies* definidas por operação, indicando o tipo e condição em que se aplicam.
Capabilities	Ativação de funcionalidades do *channel* na *blockchain*, declaradas no início do ficheiro (Código 20) (Dependem da versão do HLF).

Por último, a Código 21 mostra a secção final do ficheiro “configtx.yaml”. A secção “Profiles” é útil para os administradores definirem diferentes topologias para a rede. Note-se que para esta secção não é apresentada uma tabela com a descrição das propriedades, pois estas são variáveis.

```
Profiles:
  CarShareOrdererGenesis:
      <<: *ChannelDefaults
      Orderer:
          <<: *OrdererDefaults
          Organizations:
              - <<: *Orderer
      Consortiums:
        CarShareConsortium:
            Organizations:
                  - <<: *Gocar
                  - <<: *Drivenowrentals
                  - <<: *Rentadrive
      Application:
            <<: *ApplicationDefaults
            Organizations:
                - <<: *Gocar
                - <<: *Drivenowrentals
                - <<: *Rentadrive

  CarShareChannel:
    <<: *ChannelDefaults
    Consortium: CarShareConsortium
    Application:
        <<: *ApplicationDefaults
        Organizations:
            - <<: *Gocar
            - <<: *Drivenowrentals
            - <<: *Rentadrive
```

Código 21 - Secção "Profiles" do ficheiro "configtx.yaml".

Anexo 7 Ficheiro de configuração do Orderer

O Código 22 apresenta o ficheiro "orderer.yaml", com as definições consideradas na configuração do *ordering service* (componente *orderer*).

```
General:
    BootstrapMethod: file
    BootstrapFile: ./carshare-genesis.block
    BCCSP:
        Default: SW
        SW:
            HASH: SHA2
            Security: 256
            # Using the default 'LocalMSPDir/keystore'
            FileKeyStore:
                Keystore:
    LocalMSPDir: ./poc.hlf2.tmdei/ca/client/orderer/orderer/msp
    LocalMSPID: OrdererMSP
    ListenAddress: 127.0.0.1
    ListenPort: 7050
    Cluster:
        SendBufferSize: 10
    Keepalive:
        ServerMinInterval: 60s
        ServerInterval: 7200s
        ServerTimeout: 20s
    TLS:
        Enabled: false
FileLedger:
    Location: ./poc.hlf2.tmdei/orderer/ledger
    Prefix: hyperledger-fabric-ordererledger
Debug:
    BroadcastTraceDir:
  DeliverTraceDir:
Metrics:
    Provider: disabled
Operations:
    ListenAddress: 127.0.0.1:8443
    TLS:
        Enabled: false
Consensus:
    WALDir: /var/hyperledger/production/orderer/etcdraft/wal
    SnapDir: /var/hyperledger/production/orderer/etcdraft/snapshot
```

Código 22 - Ficheiro de configuração "orderer.yaml" utilizado para o *ordering service*.

Por sua vez, a Tabela 37 fornece a descrição das principais secções do ficheiro de configuração "orderer.yaml".

Tabela 37 - Descrição das secções e propriedades do ficheiro "orderer.yaml".

Secção	Propriedade	Descrição
General	-	Secção das propriedades gerais do *Orderer*.
	BootstrapMethod	Tipo de inicialização do componente ("file").
	BootstrapFile	Caminho para o bloco *genesis*, imprescindível para iniciar o *orderer* ("carshare-genesis.block").
	BCCSP	Serviço de encriptação utilizado pelo *Blockchain Crypto Service Provider*. Caracterização do serviço configurado, neste caso, SHA256.
	LocalMSPDir	Localização do MSP configurado previamente para o *orderer*.
	LocalMSPID	Nome único do MSP do *Orderer*.
	Cluster	Utilizado na configuração do *ordering service* em forma de *cluster*, na utilização de RAFT.
	Keepalive	Definições dos tempos, em segundos, para a disponibilidade do *orderer*.
	TLS	Configuração de servidor de *Transport Layer Security*, neste caso, inativo ("Enable: false").
FileLedger	*Location*	Localização do *ledgers* onde são persistidos os blocos de transações.
Debug	-	Definições de *debug* do *orderer*, com localização dos ficheiros de *trace* gerados pela execução.
Metrics	-	Configuração de serviço para a recolha de métricas da execução do componente, por exemplo, Prometheus. Não utilizado nesta configuração ("Provider: disabled").
Operations	*ListenAddress*	Configuração do Fabric Operations Console, que permite gerir a rede HLF através de uma interface UI.
Consensus	-	Propriedade utilizadas para o ambiente de produção, na utilização de RAFT.

Anexo 8 Configuração do Channel

O Código 23 apresentado abaixo mostra o *script* "channel-script.sh", com as funções desenvolvidas para agilizar a criação do CarShareChannel. No seguimento da configuração da rede no ambiente HLF, este *script* deve ser executado após a inicialização do *ordering service* (*script* "orderer-config.sh" do Anexo 5).

```
PEER_PATH=./poc.hlf2.tmdei/peer
ORDERER_PATH=./poc.hlf2.tmdei/orderer
CA_CLIENT_PATH=./poc.hlf2.tmdei/ca/client
CHANNEL_TRANSACTION_FILE_PATH=./poc.hlf2.tmdei/orderer/carshare-channel.tx

# Create the consortium channel (CarShareChannel)
function createChannel {
    export FABRIC_LOGGING_SPEC=INFO
    export FABRIC_CFG_PATH=$ORDERER_PATH
    configtxgen -profile CarShareChannel
        -outputCreateChannelTx $ORDERER_PATH/carshare-channel.tx
        -channelID carsharechannel
}

# Sign Carshare channel transaction file with admins users
# Use of command peerconfigtxgen
function signChannelTransactionByOrganizationAdmins {
    export FABRIC_CFG_PATH=$PEER_PATH/gocar
    export CORE_PEER_MSPCONFIGPATH=$CA_CLIENT_PATH/gocar/admin/msp
    export CORE_PEER_LOCALMSPID="GocarMSP"
    peer channel signconfigtx -f $ORDERER_PATH/carshare-channel.tx
}

# Submit the Channel creation transaction
# Use of command peer configtxgen (NEED ORDERER IN EXECUTION)
function submitChannelCreationTransaction {
    export FABRIC_CFG_PATH=$PEER_PATH/gocar
  export CORE_PEER_MSPCONFIGPATH=$CA_CLIENT_PATH/gocar/admin/msp
    peer channel create -o "localhost:7050"
        -c carsharechannel -f $CHANNEL_TRANSACTION_FILE_PATH
}

# Export channel transaction file as Json - Only for readability
function inspectChannelTransactionFile {
    mkdir -p $PEER_PATH/temp
    configtxgen -inspectChannelCreateTx $ORDERER_PATH/carshare-channel.tx >
        $PEER_PATH/temp/carshare-channel.json
}
```

Código 23 - Código do script "channel-config.sh".

Por conseguinte, na Tabela 38 encontra-se brevemente descrito as funções apresentadas acima.

Tabela 38 - Descrição da funcionalidade das funções do script "channel-config.sh".

Função	Descrição
createChannel	Função para criar o *channel* do consórcio, com o nome "carsharechannel". Como resultado da sua execução é criada uma transação que necessita ser submetida.
signChannelTransactionByOrganizationAdmins	Assinatura do ficheiro da transação de criação do *channel* ("carshare-channel.tx") pelos administradores das organizações.
submitChannelCreationTransaction	Submissão da transação do *channel* anteriormente criada, de forma a efetivar a criação do mesmo.
inspectChannelTransactionFile	Função auxiliar para converter o ficheiro da transação (formato ".tx") de criação do *channel* num ficheiro do formato JSON.

Anexo 9 Configuração dos Peers

O Código 24 apresenta o script desenvolvido para a configuração dos componentes *peers* das várias organizações integrantes na rede HLF.

```
PEER_PATH=./poc.hlf2.tmdei/peer
CA_CLIENT_PATH=./poc.hlf2.tmdei/ca/client
GOCAR_CA_CLIENT=./poc.hlf2.tmdei/ca/client/gocar
LEDGERS_PATH=./poc.hlf2.tmdei/ledgers

# Register organizations Anchors Peer's Identity to all Organizations
function registerPeers {
    export FABRIC_CA_CLIENT_HOME=$GOCAR_CA_CLIENT/admin
    fabric-ca-client register --id.type peer --id.name peer1.gocar
        --id.secret pw --id.affiliation gocar
}

# Enroll organizations Anchors Peer's Identity to all Organizations
# Add certificate of organization admin (client/ORG/admin/msp/admincerts)
function enrollPeers {
    export FABRIC_CA_CLIENT_HOME=$GOCAR_CA_CLIENT/peer1.gocar
    fabric-ca-client enroll -u http://peer1.gocar:pw@localhost:7054
    mkdir -p $GOCAR_CA_CLIENT/peer1.gocar/msp/admincerts
    cp $GOCAR_CA_CLIENT/admin/msp/signcerts/*
        $GOCAR_CA_CLIENT/peer1.gocar/msp/admincerts
}

# Start Anchor Peer1 of GoCar organization - Join CarShare channel
function startPeer1GocarAndJoinChannel {
    export FABRIC_CFG_PATH=$PEER_PATH/gocar/peer1.gocar
    export PEER_LOGS=$PEER_PATH/gocar/peer1.gocar
    mkdir -p $PEER_LOGS
    export CORE_PEER_ID=peer1.gocar
    export CORE_PEER_LOCALMSPID="GocarMSP"
    export CORE_PEER_MSPCONFIGPATH=$GOCAR_CA_CLIENT/peer1.gocar/msp
    export CORE_PEER_FILESYSTEMPATH=$LEDGERS_PATH/gocar/peer1.gocar/ledger
    sudo -E mkdir -p $CORE_PEER_FILESYSTEMPATH
    # Start peer - Fetch and join CarShare channel
    sudo -E peer node start 2> $PEER_LOGS/peer.log &
    export CORE_PEER_MSPCONFIGPATH=$GOCAR_CA_CLIENT/admin/msp
    peer channel fetch 0 ./carsharechannel.block -o localhost:7050
        -c carsharechannel
    peer channel join -o localhost:7050 -b ./carsharechannel.block
}
```

Código 24 - Código do script "peer-config.sh".

Note-se, no código anterior (Código 24), atendendo à similaridade do processo, apenas é mostrado código para a configuração de um *peer* da Gocar, sendo necessário ligeira adaptação para outras organizações.

A Tabela 39 descreve as funções do código (Código 24).

Tabela 39 - Descrição da funcionalidade das funções do script "peer-config.sh".

Função	Descrição
registerPeers	Registo da *identity* no CA *server* de todos os *peers* (inicialmente propostos) para as várias organizações, com o comando “fabric-ca-client register”.
enrollPeers	*Enroll* dos vários *peers* das várias organizações, com o comando “fabric-ca-client enroll”.
startPeer1GocarAndJoinChannel	Criação do diretório afeto ao *peer*, execução nas portas definidas e entrada no mesmo no *channel* do consórcio (“carsharechannel”), com os comandos “peer channel fetch” e “peer channel join”.

Anexo 10 Ficheiro de Configuração do componente Peer

O ficheiro "core.yaml", abaixo no Código 25 e Código 26, mostra as configurações definidas para o “peer1.gocar” da organização Gocar. Note-se que algumas propriedades foram ocultadas por não serem utilizados no contexto do trabalho descrito neste documento. Para os *peers* das restantes organizações foram utilizadas configurações equivalentes.

```
peer:
    id: peer1.gocar
    listenAddress: 0.0.0.0:7051
    address: 0.0.0.0:7051
    addressAutoDetect: false
    keepalive:
        minInterval: 60s
        client:
            interval: 60s
            timeout: 20s
        deliveryClient:
            interval: 60s
            timeout: 20s
    gossip:
        bootstrap: 127.0.0.1:7051
        useLeaderElection: true
        orgLeader: false
    tls:
        enabled:  false
    fileSystemPath: ./poc.hlf2.tmdei/ledgers/gocar/ledger
    BCCSP:
        Default: SW
        SW:
            Hash: SHA2
            Security: 256
            FileKeyStore:
                KeyStore:
    mspConfigPath: ./poc.hlf2.tmdei/ca/client/gocar/admin/msp
    localMspId: GocarMSP
    localMspType: bccsp
    discovery:
        enabled: true
```

Código 25 – Parte um do ficheiro de configuração "core.yaml" utilizado na definicão do *peer* “peer1.gocar”.

```
chaincode:
    id:
        path:
        name:
    pull: false
    golang:
        runtime: $(DOCKER_NS)/fabric-baseos:$(TWO_DIGIT_VERSION)
        dynamicLink: false
    installTimeout: 300s
    startupTimeout: 300s
    executeTimeout: 30s
    mode: net
    keepalive: 0
    system:
        _lifecycle: enable
        cscc: enable
        lscc: enable
        escc: enable
        vscc: enable
        qscc: enable
    logging:
      level:  info
      shim:   warning
      format: '%{color}%{time:2006-01-02 15:04:05.000 MST} [%{module}]
%{shortfunc} -> %{level:.4s} %{id:03x}%{color:reset} %{message}'
ledger:
  state:
    stateDatabase: goleveldb
    totalQueryLimit: 100000
 history:
    enableHistoryDatabase: true
operations:
    listenAddress: 127.0.0.1:9443
metrics:
    provider: disabled
```

Código 26 - Parte dois do ficheiro de configuração "core.yaml" utilizado na definicão do peer “peer1.gocar”.

A Tabela 40 descreve resumidamente as secções do ficheiro de configuração do *peer*.

Tabela 40 - Descrição das secções e propriedades do ficheiro "core.yaml".

Secção	Propriedade	Descrição
peer	-	Definições gerais do *Peer*.
	id	Identificador/nome único do componente na rede.
	listenAddress	Endereço de escuta com porta do *Peer* na rede.
	address	Representa o *endpoint* do *Peer*.
	addressAutoDetect	Quando ativo, em ambiente de Docker *containers*, o endereço do *Peer* é automaticamente determinado.
	keepAlive	Definições dos tempos, em segundos, para disponibilidade do *Peer*.
	gossip	Configurações gerais do protocolo GOSSIP, como o endereço de *bootstrap* e atribuição de *Peer lead* da organização.
	tls	Configuração de servidor de Transport Laver Security, (inativo - "Enable: false").
	fileSystemPath	Caminho para a localização onde o *Peer* guardará informação, por exemplo, o *ledger*.
	BCCSP	Serviço de encriptação utilizado pelo Blockchain Crypto Service Provider, caracterização do serviço configurado (SHA256).
	mspConfigPath	Caminho para a localização do MSP configurado previamente para o *Peer.*
	localMspId	Identificador/Nome único do MSP do *Peer*.
	localMspType	Tipo de MSP configurado (“bccsp”).
	discovery	Definições do serviço para consultar informação relativa ao *Peer*.
chaincode	-	Definições dos *chaincodes* a instalar/executar no *Peer*.
	id	Identificador/nome único utilizado pela interface *ChaincodeStub* para o *chaincode*.
	pull	Indicador da utilização do mecanismo *pull* de *images* Docker dos *chaincodes*.
	golang	Definições gerais da linguagem GO.
	installTimeout	Tempo limite para a instalação do *chaincode* no *Peer*.
	startupTimeout	Tempo limite definido para a inicialização do *chaincode*.
	executeTimeout	Tempo limite para a execução do *chaincode*.
	system	Listagem dos *chaincodes* de sistema.
	logging	Definições gerais dos logs dos *chaincodes*, nível (“info”) e formato.
ledger	-	Definições do *ledger* respetivo ao *Peer*.
	state	Configuração da base de dados do *state*, com indicação do tipo de tecnologia a utilizar (LevelDB).
	history	Ativação de persistência de histórico dos valores do *state*.
operations	-	Configuração do Fabric Operations Console, que permite gerir a rede HLF através de uma interface UI.
metrics	-	Configuração de serviço para a recolha de métricas da execução do componente, por exemplo, Prometheus.

Anexo 11 Ficheiro docker-compose

O ficheiro *docker-compose.yml* do Código 27 e Código 28 apresenta as configurações dos containers "orderer.com" e "peer1.gocar.com", correspondentes aos componentes *orderer* e *peer1*, respetivamente. De notar que foram omitidas as configurações dos *peers* de outras organizações dada a similaridade com o "peer1.gocar.com".

```
version: '3'
networks:
  poc:
volumes:
  data-orderer:
  data-peer1.gocar:
services:
  orderer.com:
    container_name: orderer.com
    image: hyperledger/fabric-orderer:2.5.1
    environment:
      - FABRIC_LOGGING_SPEC=info
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_GENESISMETHOD=file
      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/genesis.block
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
    command: orderer
    volumes:
      - ./poc.hlf2.tmdei/orderer/genesis.block:
            /var/hyperledger/orderer/genesis.block
      - ./poc.hlf2.tmdei/ca/client/orderer/admin/msp:
            /var/hyperledger/orderer/msp
      - ./poc.hlf2.tmdei/orderer:/var/hyperledger/orderer
      - ./poc.hlf2.tmdei/chaincodes:/etc/hyperledger/fabric/chaincodes
    ports:
      - 7050:7050
    networks:
      - poc
```

Código 27 - Parte um do ficheiro "docker-compose.yml".

```
peer1.gocar.com:
    container_name: peer1.gocar.com
    image: hyperledger/fabric-peer:2.5.1
    depends_on:
      - orderer.com
    environment:
      - FABRIC_LOGGING_SPEC=debug
      - CORE_PEER_LOCALMSPID=GocarMSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp
      - CORE_PEER_ID=peer1.gocar.com
      - CORE_PEER_ADDRESS=peer1.gocar.com:7051
      - CORE_PEER_TLS_ENABLED=false
      - CORE_CHAINCODE_LOGLEVEL=debug
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docker_poc
    command: peer node start --peer-chaincodedev=true
    volumes:
      - /var/run/:/host/var/run/
      - ./poc.hlf2.tmdei/ca/client/gocar/admin/msp:
            /etc/hyperledger/fabric/msp
      - /poc.hlf2.tmdei/ledgers/peer1.gocar.com:
            /etc/hyperledger/fabric/ledger
      - ./poc.hlf2.tmdei/orderer:/var/hyperledger/orderer
      - /./poc.hlf2.tmdei/chaincodes:/etc/hyperledger/fabric/chaincodes
     ports:
      - 7051:7051
    networks:
     - poc
```

Código 28 – Parte dois do ficheiro “docker-compose.yml”.

Após a execução do ficheiro apresentado anteriormente com o comando “docker-compose up”, é possível ver os *containers* criados e em execução através do Docker Desktop, demonstrado na Figura 38.

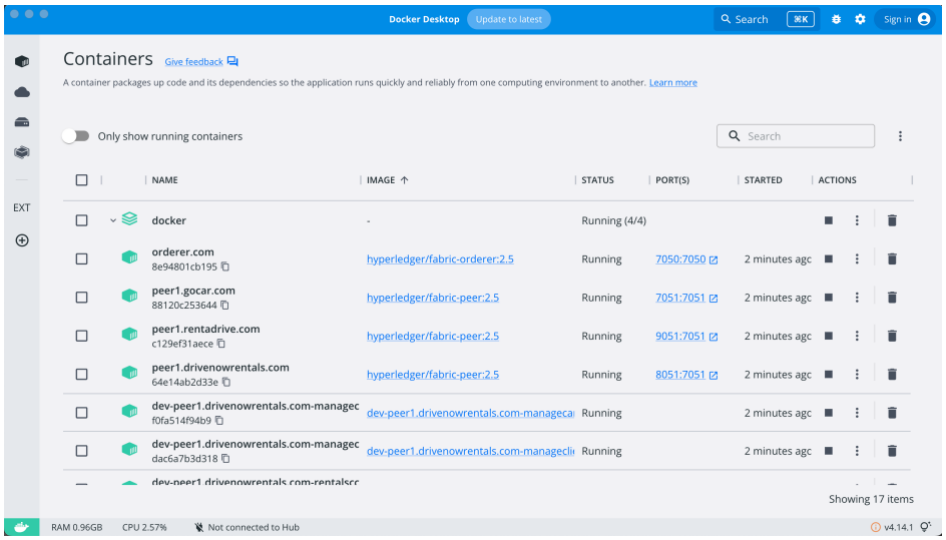


Figura 38 – Visualização dos *containers* no Docker Desktop

Anexo 12 Implementação dos Smart Contracts do ManageCarCC

Este anexo apresenta a implementação dos *smart contracts* mais relevantes do *chaincode ManageCarCC*, anteriormente descritos na Tabela 21 da secção 6.2. O Código 29 é a função auxiliar "*ValidateCarProposal*", que é invocada na função "*CreateCarProposal*" (Código 4) com o propósito de validar dos dados de *input* para o registo de um novo automóvel.

```
func ValidateCarProposal(
    category string,
    company string,
    fuel string,
    licensePlate string,
    seats int) string {

    var validationMessage string
    if GetCategory(category) == NoneCategory {
       validationMessage += "invalid car category "
    }

    if GetCompany(company) == Default {
       validationMessage += "invalid rent company "
    }

    if GetFuel(fuel) == NoneFuel {
       validationMessage += "invalid fuel type "
    }

    if !regexp.MustCompile(LicensePlateRegexPattern).MatchString(licensePlate)
    {
       validationMessage += "invalid license plate format (e.g. 'AB-12-12') "
    }

    if seats <= 0 {
       validationMessage += "invalid number of seats"
    }

    return validationMessage
}
```

Código 29 - Função auxiliar "ValidateCarProposal" implementada no chaincode *ManageCarCC*.

Por sua vez, o *smart contract* do Código 30 é particularmente útil na implementação do *chaincode SharingAgreementsCC*.

```
func (s *SmartContract) GetCarOwnerCompanyByLicensePlate(
    ctx contractapi.TransactionContextInterface,
    licensePlate string) (Company, error) {

    existingCar, err := s.GetCarByLicensePlate(ctx, licensePlate)
    return existingCar.OwnerCompany, err
}
```

Código 30 - *Smart contract* "*GetCarOwnerCompanyByLicensePlate*" implementado no *chaincode ManageCarCC*.

Como mencionado pela descrição do *chaincode ManageCarCC* (secção 6.2), o *smart contract* abaixo no Código 31 invoca a função auxiliar "*CreateCarProposal*", para instanciar um novo *Car* e de seguida guarda no *state*, com recurso à função "*PutState*".

```
func (s *SmartContract) CreateCar(ctx contractapi.TransactionContextInterface, brand string, category string,
    company string, fuel string, licensePlate string, seats int) (*Car, error) {

    existingCar, err := s.GetCarByLicensePlate(ctx, licensePlate)
    if existingCar != nil {
       return existingCar, fmt.Errorf("the car asset with license plate %s already exists", licensePlate)
    }

    carProposal, validationMessage := CreateCarProposal(
       brand,
       category,
       company,
       fuel,
       licensePlate,
       seats)

    if validationMessage != "" {
       return nil, fmt.Errorf(validationMessage)
    }

    carJson, err := json.Marshal(carProposal)
    if err != nil {
       return nil, fmt.Errorf(err.Error())
    }

    err = ctx.GetStub().PutState(carProposal.LicensePlate, carJson)
    if err != nil {
       return nil, fmt.Errorf("failed to put new car asset to world state")
    }

    createdCar, err := s.GetCarByLicensePlate(ctx, licensePlate)
    return createdCar, nil
}
```

Código 31 - *Smart contract* "*CreateCar*" implementado no *chaincode ManageCarCC*.

Anexo 13 Implementação dos Smart Contracts do SharingAgreementsCC

Neste anexo, abaixo no Código 32 é mostrado a função "*GetSharingAgreementByCompanies*" (descrita na Tabela 22) constituinte do *chaincode SharingAgreementsCC*. Este *smart contract* apresenta especial relevância na validação da criação de um novo *Rental* numa empresa que tente alugar um automóvel que não é sua propriedade, uma vez que retorna o acordo de partilha ativo entre duas empresas ("firstCompany" e "secondCompany").

```
func (s *SmartContract) GetSharingAgreementByCompanies(
    ctx contractapi.TransactionContextInterface,
    firstCompany string,
    secondCompany string) (*SharingAgreement, error) {

    if firstCompany == "" || secondCompany == "" {
       return nil, fmt.Errorf("provided companies are invalid")
    }

    existingAgreementJson, err :=
        ctx.GetStub().GetState(
            CreateSharingAgreementId(firstCompany, secondCompany))

    if existingAgreementJson == nil {
       existingAgreementJson, err =
           ctx.GetStub().GetState(
               CreateSharingAgreementId(secondCompany, firstCompany))
    }

    if err != nil {
       return nil, fmt.Errorf("failed to read from world state")
    }

    if existingAgreementJson == nil {
       return nil, fmt.Errorf("agreement with provided id doesn't exist")
    }

    var existingAgreement SharingAgreement
    err = json.Unmarshal(existingAgreementJson, &existingAgreement)
    if err != nil {
       return nil, err
    }

    if existingAgreement.IsActive {
       return &existingAgreement, nil
    }

    return nil, nil
}
```

Código 32 - *Smart contract* "*GetSharingAgreementByCompanies*" implementado no *chaincode SharingAgreementsCC.*

Anexo 14 Implementação dos Smart Contracts do RentalsCC

Este anexo apresenta a função auxiliar "*ValidateRentalProposal*" (Código 33) e o *smart contract* "*DeliverRentalCar*" (Código 34).

```
func ValidateRentalProposal(ctx contractapi.TransactionContextInterface,
    carLicensePlate string, clientId string, company string) (string, string)
{
    var validationMessage string
    queryArgs := SetQueryParams([]string{"GetClientById", clientId})
    clientResponse := ctx.GetStub().InvokeChaincode(
       CLIENTS_CHAINCODE, queryArgs, CARSHARECHANNEL)
    if clientResponse.Status != shim.OK {
       validationMessage += "failed to get client "
    } else if string(clientResponse.Payload) == "" {
       validationMessage += "client with id provided not found"
    }

    queryArgs = SetQueryParams(
       []string{"GetCarOwnerCompanyByLicensePlate", carLicensePlate})
    carResponse := ctx.GetStub().InvokeChaincode(
       CARS_CHAINCODE, queryArgs, CARSHARECHANNEL)
    if carResponse.Status != shim.OK {
       validationMessage += "failed to get car "
    } else if string(carResponse.Payload) == "" {
       validationMessage += "car with license plate provided not found "
    }

    carCompany := strings.ToLower(string(carResponse.Payload))
    rentalCompany := strings.ToLower(company)
    if carCompany != rentalCompany {
       queryArgs = SetQueryParams([]string{
           "GetSharingAgreementByCompanies", carCompany, rentalCompany})
       sharingAggResponse := ctx.GetStub().InvokeChaincode(
          SHARINGAGREEMENT_CHAINCODE, queryArgs, CARSHARECHANNEL)
       if sharingAggResponse.Status != shim.OK {
          validationMessage +="agreement for provided companies doesn't exist"
       }
    }

    if GetCompany(company) == Default {
       validationMessage += "invalid rent company "
    }
    return validationMessage, carCompany
}
```

Código 33 - *Smart contract* "*ValidateRentalProposal*" implementado no *chaincode RentalsCC*.

A função acima (Código 33) é utilizada no *smart contract* "*CreateRental*" explorada anteriormente no diagrama de sequência da Figura 33. Esta é fundamental para a criação de um novo *rental*, pois detém a responsabilidade de invocar outros *smart contracts* de outros *chaincodes*, para validar os dados de *input* da transação ("carLicensePlate", "clientId" e *sharing agreement*).

O *smart contract* abaixo no Código 34 termina o aluguer de um automóvel, alterando o estado do *rental* para "Finished", e invoca o *smart contract* "*ReleaseCarByLicensePlate*" do *chaincode ManageCarCC* para alterar o estado do automóvel para "Available". Além disso, determina as "Penalties" a aplicar no *rental* conforme descrito no final da descrição do *chaincode RentalsCC* (secção 6.2).

```
func (s *SmartContract) DeliverRentalCar(ctx contractapi.TransactionContextInterface, rentalId string, usedDistance int) error {

    existingRental, err := s.GetRentalById(ctx, rentalId)
    if err != nil {
       return err
    }

    if existingRental.Status == Inactive {
       return fmt.Errorf("the rental with id %s is inactive", rentalId)
    }

    if existingRental.Status == Finished {
       return fmt.Errorf("the rental with id %s it's already finished", rentalId)
    }

    existingRental.DeliveryDate, _ = time.Parse(time.DateTime, time.Now().Format(time.DateTime))
    existingRental.DeliveryDistance = usedDistance
    existingRental.Status = Finished

    if existingRental.CarOwnerCompany != existingRental.Company {
       queryArgs := SetQueryParams([]string{"GetSharingAgreementByCompanies", string(existingRental.CarOwnerCompany),
           string(existingRental.Company)})

       sharingAgResponse := ctx.GetStub().InvokeChaincode(SHARINGAGREEMENT_CHAINCODE, queryArgs, CARSHARECHANNEL)

       if sharingAgResponse.Status != shim.OK {
          return fmt.Errorf("failed on get Sharing Agreement. Details: %s", sharingAgResponse.GetMessage())
       }

       var sharingAgreement SharingAgreement
       err = json.Unmarshal(sharingAgResponse.Payload, &sharingAgreement)
       if err != nil {
          return err
       }
```

```
        totalDuration := existingRental.StartDate.AddDate(0, 0, sharingAgreement.DurationCondition.LimitDurationValue)
        if existingRental.DeliveryDate.After(totalDuration) {
            existingRental.Penalties = append(existingRental.Penalties, Penalty{
                Description: "Penalty applied: Vehicle delivery deadline exceeded.",
                Value:       sharingAgreement.DurationCondition.PenaltyValue,
            })
        }

        if usedDistance > sharingAgreement.DistanceCondition.LimitDistanceValue {
            existingRental.Penalties = append(existingRental.Penalties, Penalty{
                Description: "Penalty applied: Distance traveled in the vehicle exceeded the limit.",
                Value:       sharingAgreement.DistanceCondition.PenaltyValue,
            })
        }
    }

    rentalJson, err := json.Marshal(existingRental)
    if err != nil {
        return err
    }

    queryArgs := SetQueryParams([]string{"ReleaseCarByLicensePlate", existingRental.CarLicensePlate})
    carResponse := ctx.GetStub().InvokeChaincode(CARS_CHAINCODE, queryArgs, CARSHARECHANNEL)
    if carResponse.Status != shim.OK {
        return fmt.Errorf("failed changing car status to AVAILABLE: %d ", carResponse.Status)
    }

    return ctx.GetStub().PutState(rentalId, rentalJson)
}
```

Código 34 - *Smart contract* "*DeliverRentalCar*" implementado no *chaincode RentalsCC*.

Anexo 15 Chaincode Lifecycle

Neste anexo, são apresentados os vários comandos do *Fabric chaincode lifecycle* utilizados para a instalação/atualização dos *chaincodes* nos *peers* da BC. O código abaixo (Código 35) exemplifica os vários passos para a instalação do *RentalsCC* no Peer1 da Gocar, descritos previamente no final da secção 6.2 do documento.

```
export SEQUENCE=8
export CC_LABEL=rentalscc-1.7
export CC_VERSION=1.7
export CORE_PEER_LOCALMSPID="GocarMSP"
export FABRIC_CFG_PATH=./poc.hlf2.tmdei/peer/gocar/peer1.gocar
export CORE_PEER_MSPCONFIGPATH=./poc.hlf2.tmdei/ca/client/gocar/admin/msp

# 1. Packaging
peer lifecycle chaincode package rentalscc.tar.gz
       --path ./poc.hlf2.tmdei/chaincodes/rentals
       --lang golang
       --label $CC_LABEL

# 2. Installation in Peers
peer lifecycle chaincode install ./chaincodes/rentalscc.tar.gz
peer lifecycle chaincode queryinstalled

# 3. Approve by Organizations
peer lifecycle chaincode approveformyorg
       -o 127.0.0.1:7050
       --channelID carsharechannel
       --package-id $CC_PACKAGE_ID
       --name rentalscc
       --version $CC_VERSION
       --sequence $SEQUENCE
       --signature-policy "OR('GocarMSP.member','DrivenowrentalsMSP.member')"

# 4. Commit
peer lifecycle chaincode commit
       -o 127.0.0.1:7050
       --channelID carsharechannel
       --name rentalscc
       --version $CC_VERSION
       --sequence $SEQUENCE
       --signature-policy "OR('GocarMSP.member','DrivenowrentalsMSP.member')"
       --peerAddresses 127.0.0.1:7051 127.0.0.1:8051 127.0.0.1:9051

peer lifecycle chaincode querycommited
       --channelID carsharechannel
       --name rentalscc
```

Código 35 – Comandos do *chaincode lifecycle*, utilizados na instalação de um *chaincode*.

Anexo 16 API Swagger dos Chaincodes

Na seguinte figura (Figura 39) é apresentada a interface da documentação Swagger implementada na *web* REST API, anteriormente descrita (secção 6.2 e Tabela 24). Esta interface permite invocar os *endpoints*, enviando os parâmetros e *payloads* requeridos pelos *smart contracts*.

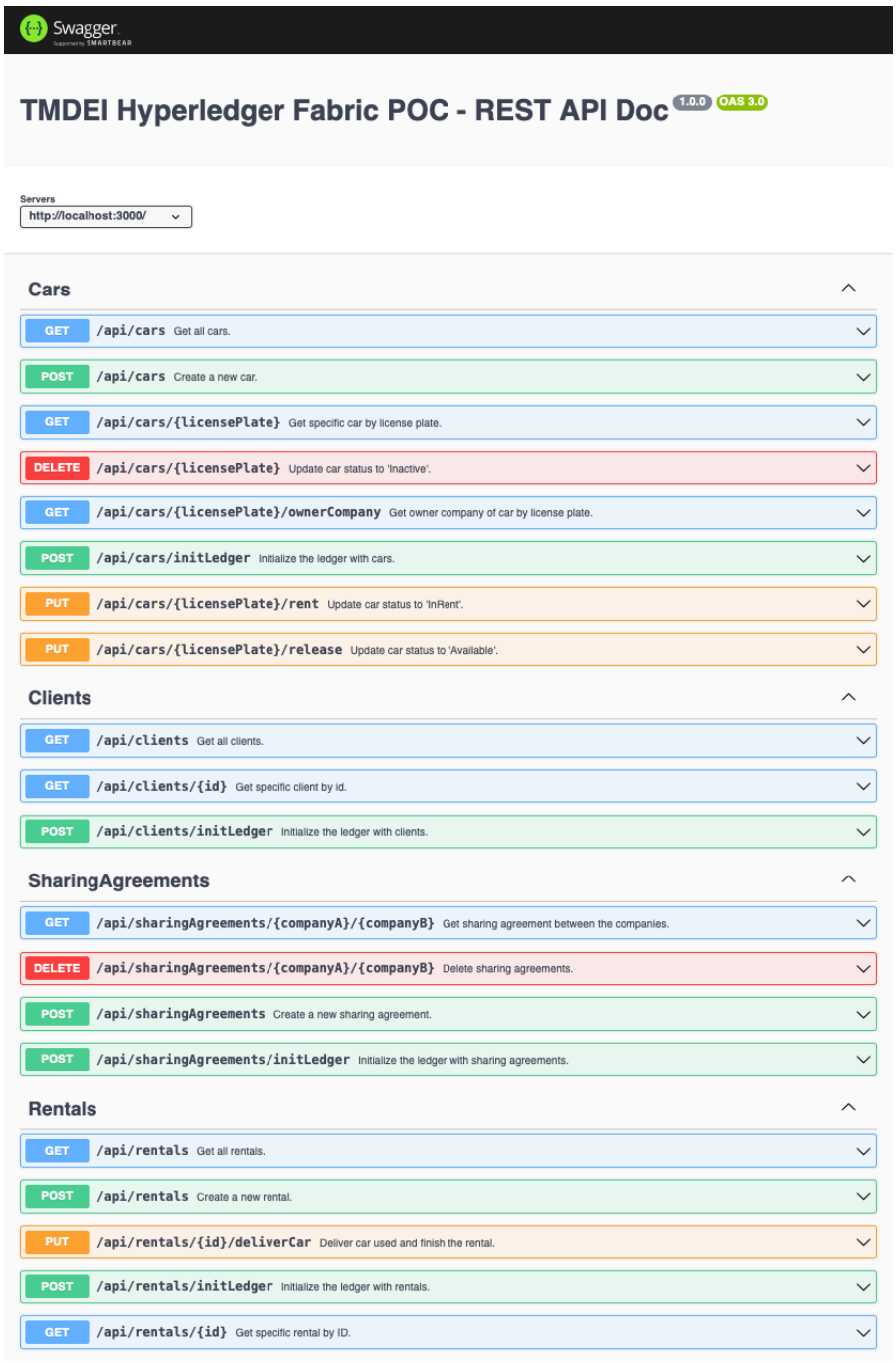


Figura 39 - Interface Swagger da REST API desenvolvida para invocar os *smart contracts*.

Anexo 17 Testes Unitários

Este anexo pretende adicionar mais contexto relativos aos testes unitários desenvolvidos (secção 7.1). O Código 36 abaixo apresenta um teste unitário desenvolvido no âmbito do *RentalsCC*. De notar que, devido à dependência entre *chaincodes* e de um *ledgers* funcional, nos testes unitários são utilizadas interfaces *mock* que visam simular as chamadas aos *chaincodes* e *state*.

```
func TestCreate_GivenValidLicensePlateAndClientAndCompany
    _WhenInvokeManageCarCCAndNotFound_ShouldReturnExpectedErrors(t *testing.T){
    // Arrange
    chaincodeStub, transactionContext, rentalsCC := MockEnvironment()
    chaincodeStub.InvokeChaincodeReturnsOnCall(
        0, peer.Response{Status: shim.OK, Payload: []byte{0}})
    chaincodeStub.InvokeChaincodeReturnsOnCall(
        1, peer.Response{Status: shim.OK})
    chaincodeStub.InvokeChaincodeReturnsOnCall(
        2, peer.Response{Status: shim.OK, Payload: []byte{0}})

    rental := CreateRental(
        "AB-12-24", "1", chaincode.Gocar, chaincode.DriveNowRentals)

    // Act
    newAsset, err := rentalsCC.CreateRental(transactionContext,
       rental.CarLicensePlate, rental.ClientId, string(rental.Company))

    // Assert
    if newAsset != nil { err = fmt.Errorf("unexpected rental creation") }
    require.EqualError(t, err, "car with license plate provided not found ")
}
```

Código 36 - Teste unitário do *smart contract* "*CreateRental*" do *chaincode RentalsCC*.

As figuras Figura 40, Figura 41, Figura 42 e Figura 43 apresentam os relatórios de execução de todos os testes unitários, automaticamente gerados pelo IDE Jetbrains GoLand[26].

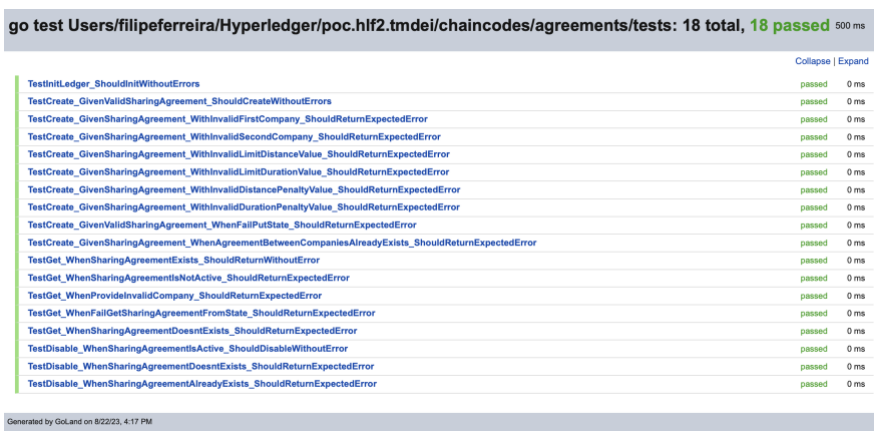


Figura 40 - Relatório de execução dos testes unitários do *chaincode SharingAgreementsCC*.

[26] Página official Jetbrains GoLand: 'jetbrains.com/go'.

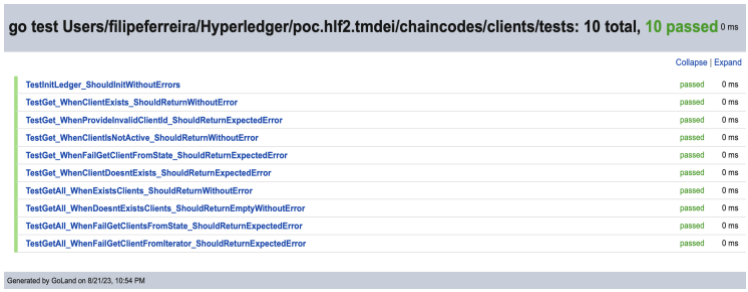


Figura 41 - Relatório de execução dos testes unitários do *chaincode ManageClientsCC*.

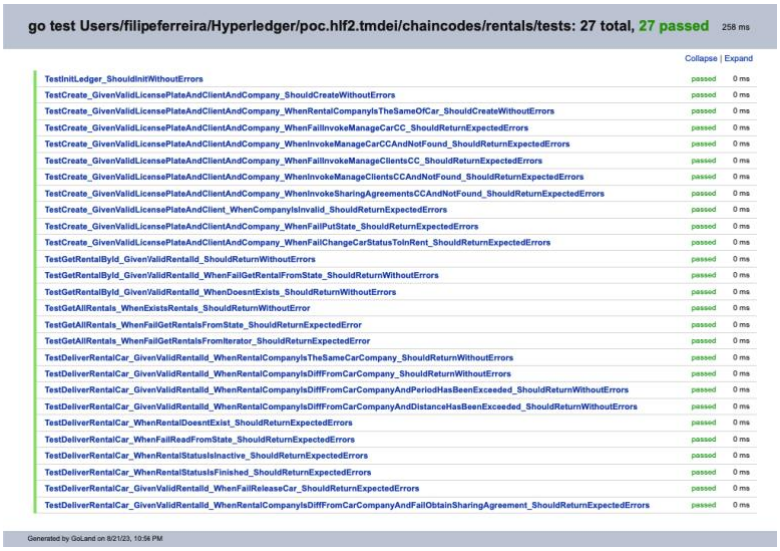


Figura 42 - Relatório de execução dos testes unitários do *chaincode RentalsCC*.

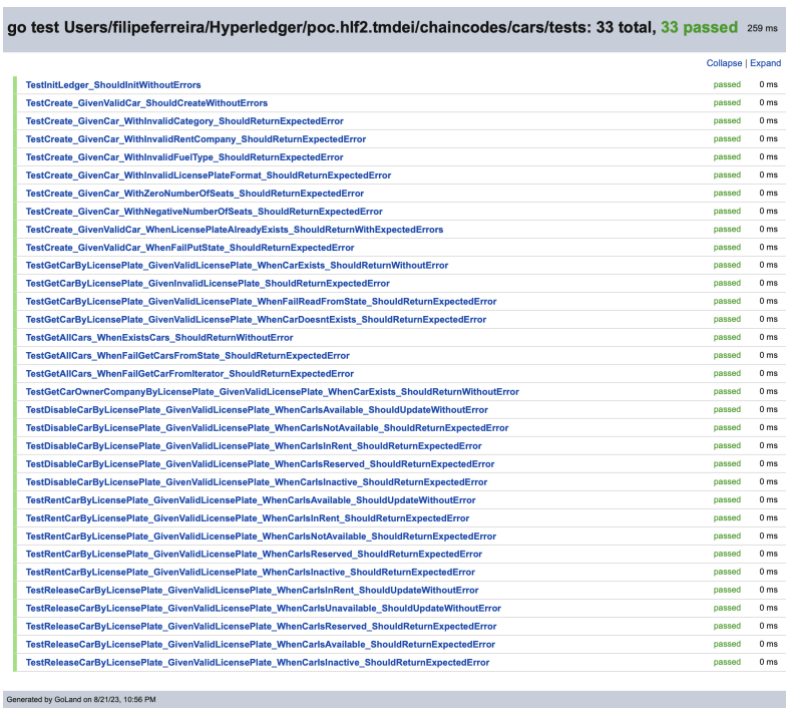


Figura 43 - Relatório de execução dos testes unitários do *chaincode ManageCarCC*.

Anexo 18 Testes de Integração Postman

Conforme anteriormente mencionado (secção 7.2), a Figura 44 mostra o resultado da execução em sequência de todos os testes de integração desenvolvidos para atender aos cenários da Tabela 26 (secção 7.2), na ferramenta Postman.

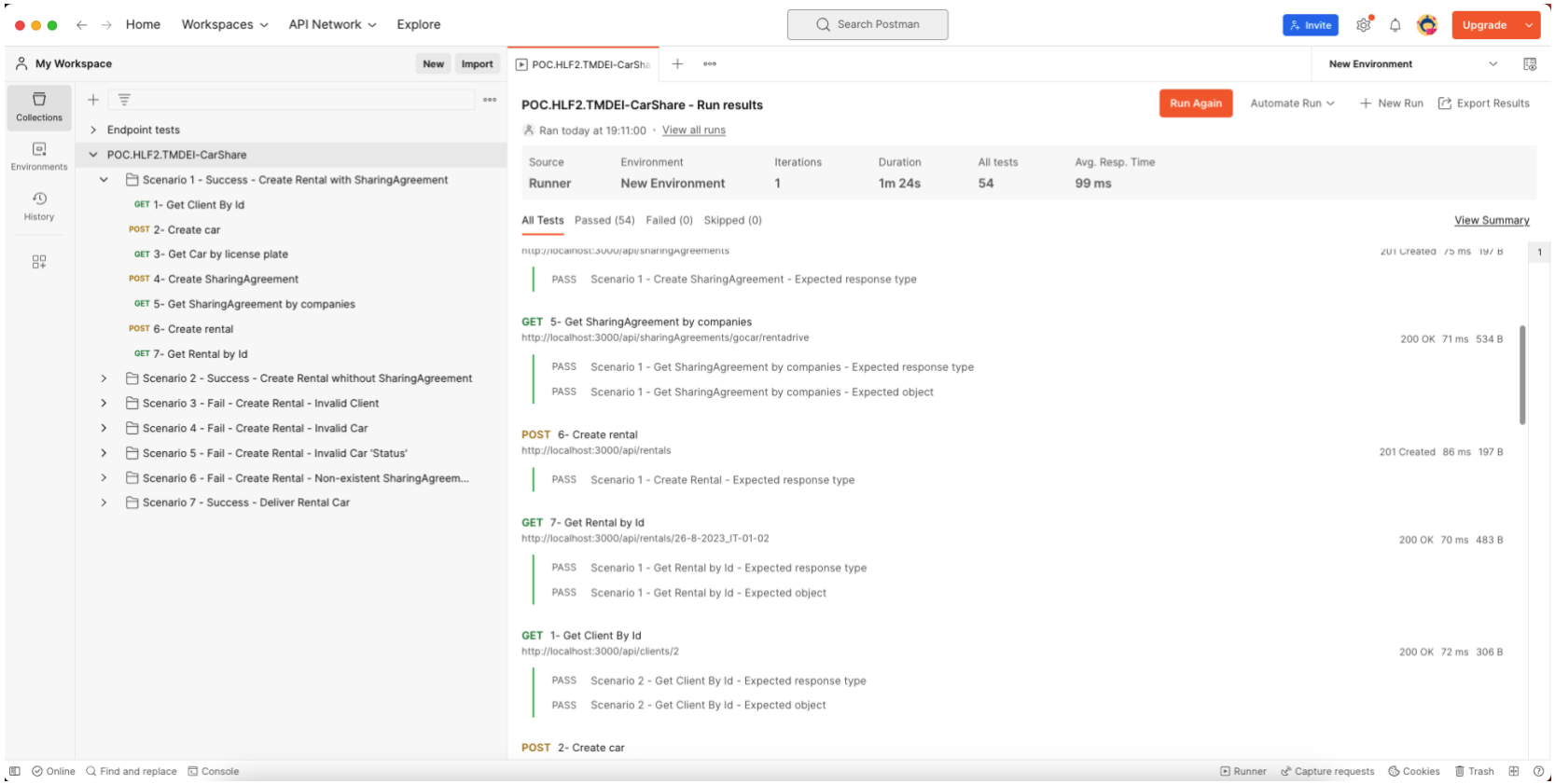


Figura 44 - Resultado da execução dos testes de integração no Postman.