UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Proteção de Identidade com Differential Privacy

Tomás Miguel Frade Piteira

Mestrado em Engenharia Informática

Trabalho de Projeto orientado por:
Prof. Doutor Pedro Jorge Fernandes Ângelo

2025

Agradecimentos

Em primeiro lugar, gostaria de expressar o meu agradecimento ao meu orientador, Prof. Dr. Pedro Ângelo, pela disponibilidade, paciência e orientação ao longo deste percurso. Todo o seu apoio constante e múltiplos conselhos foram determinantes para a concretização deste projeto.

Agradeço também à Ana Guimarães, por me ter recebido na sua empresa de forma tão acolhedora e por me proporcionar a oportunidade de entrar pela primeira vez no mundo do trabalho. Ter tido esta oportunidade de desenvolver este projeto rodeado de uma equipa de excelência foi uma experiência muito enriquecedora e uma aprendizagem que levarei para o resto da minha vida.

Quero igualmente agradecer a todos os colaboradores da Trust Systems com quem me cruzei, pela simpatia, boa disposição e disponibilidade em ajudar sempre que precisei. Desde o primeiro dia que me senti sempre muito bem recebido, o que tornou o ambiente de trabalho muito agradável e motivador. Aprendi imenso com todos e levarei comigo todo o conhecimento que me passaram.

Um agradecimento especial ao meu orientador na empresa, Artiom Andronic, por toda a disponibilidade, pelas dúvidas esclarecidas, pelas chamadas sempre que surgiam problemas, pela exigência e pelo interesse genuíno em que tudo corresse da melhor forma. Todo este apoio, especialmente na componente mais prática do projeto, foi determinante para o sucesso deste trabalho.

Finalmente, um agradecimento enorme à minha família e aos meus amigos, por estarem sempre ao meu lado, por me compreenderem e me apoiarem nos momentos mais difíceis. A simples presença, incentivo e confiança tornaram todo este percurso mais leve e possível de concretizar.

À minha Família e Amigos

Resumo

Nos últimos anos, temos assistido a uma evolução tecnológica que tem conduzido à produção e recolha de grandes quantidades de dados, que se tornaram essenciais para a tomada de decisões em diversas áreas do nosso quotidiano. No entanto, este crescimento exponencial trouxe inúmeros desafios significativos no que respeita à proteção da identidade e da privacidade dos indivíduos, uma vez que a utilização inadequada desses dados pode comprometer informações mais críticas.

O estudo apresentado neste documento tem como objetivo explorar e aplicar a técnica de Privacidade Diferencial, uma solução matemática rigorosa que permite assegurar elevados níveis de proteção a informações consideradas sensíveis. O projeto apresenta os fundamentos teóricos desta técnica, discute as suas vantagens e possíveis limitações e demonstra, sobretudo, a sua aplicação prática através do desenvolvimento de uma ferramenta de Privacidade Diferencial.

Neste contexto, o trabalho propõe, em parceria com a Trust Systems, a criação de uma ferramenta que implemente Privacidade Diferencial de forma prática e eficiente. A solução combina diferentes mecanismos desta técnica, garantindo que os dados individuais permanecem protegidos, ao mesmo tempo que se preserva a sua utilidade para análises estatísticas e tomadas de decisão.

Palavras-Chave: Privacidade, Confidencialidade, Big Data

Abstract

In recent years, we have witnessed a technological evolution that has led to the production and collection of large amounts of data, which have become essential for decision-making in various areas of our daily lives. However, this exponential growth has brought significant challenges regarding the protection of individual's identity and privacy, as the improper use of such data can compromise critical information.

The study presented in this document aims to explore and apply the technique of Differential Privacy, a rigorous mathematical solution that ensures high levels of protection for sensitive information. The project presents the theoretical foundations of this technique, discusses its advantages and potential limitations, and, above all, demonstrates its practical application through the development of a Differential Privacy tool.

In this context, the work proposes, in partnership with Trust Systems, the creation of a tool that implements Differential Privacy in a practical and efficient manner. The solution combines different mechanisms of this technique, ensuring that individual data remain protected while preserving their utility for statistical analyses and decision-making.

Keywords: Privacy, Confidentiality, Big Data

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Abreviaturas

API *Application Programming Interface*

CRM *Customer Relationship Management*

DNS *Domain Name System*

ERD *Entity Relationship Diagram*

GUI *Graphical User Interface*

HTTP *Hypertext Transfer Protocol*

HTTPS *HyperText Transfer Protocol Secure*

iOS *iPhone Operating System*

JSON *JavaScript Object Notation*

SQL *Structured Query Language*

URL *Uniform Resource Locator*

Capítulo 1

Introdução

1.1 Motivação e Problema

Nos últimos anos, temos assistido a um avanço crescente da tecnologia, marcado pela produção de grandes quantidades de dados a uma velocidade sem precedentes. Face a esta evolução digital surgiu o termo Big Data [40], um conceito que se refere à recolha, análise e processamento de grandes volumes de dados que são essenciais para todas as tomadas de decisões nas mais diversas áreas da nossa sociedade. Seja no setor empresarial, ou até mesmo em áreas como a saúde e a educação, a análise detalhada desses dados permite convertê-los em informações estratégicas.

Embora tragam inúmeros benefícios, todos esses avanços tecnológicos apresentam também desafios no que diz respeito à proteção de identidade e privacidade. Este cenário tem criado uma crescente preocupação em torno da segurança da informação, uma vez que é cada vez mais difícil controlar como os diversos dados são utilizados e até que ponto estamos ou não sujeitos à exposição de certas informações consideradas mais sensíveis. Existe, portanto, uma dicotomia muito bem definida: ou se permite a partilha de dados para fins que podem levar a grandes benefícios no futuro, ou priorizamos a garantia de privacidade das nossas informações pessoais.

Sendo um tema que tem vindo a tornar-se cada vez mais relevante nos dias de hoje, é então fundamental garantir que existem diferentes soluções e ferramentas eficazes que permitam controlar cuidadosamente até onde determinadas informações pessoais podem ser divulgadas.

Atualmente, existem várias técnicas de anonimização como a k-anonimização [45], l-diversity [32], t-closeness [31], generalização e supressão [10] , que são utilizadas para proteger a privacidade dos dados. Contudo, com o avanço das necessidades de segurança e privacidade, têm surgido novos métodos e técnicas com mais e melhores garantias no cumprimento deste objetivo. Assim, é com base nesse contexto que surge o tema principal deste projeto: Privacidade Diferencial - uma solução matemática que fornece aos investigadores e analistas de dados a possibilidade de obterem informações relevantes acerca dos indivíduos, sem nunca revelar as identidades próprias dos mesmos. Esta técnica será o objeto de estudo, sendo detalhada ao longo de todo este projeto.

1.2 Contexto e Objetivos

Este trabalho foi desenvolvido no contexto do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa, consistindo no projeto final do curso. Trata-se de uma proposta inicialmente apresentada por uma determinada empresa e posteriormente aprovada pela Faculdade, estabelecendo assim uma colaboração formal para a realização do projeto.

Esta tese foi realizada em parceria com a Trust Systems [47], uma empresa portuguesa fundada em 2016 que atua na área da Segurança de Informação. Tem como principais focos soluções de proteção de dados (cifragem e anonimização), auditorias e serviços especializados, tendo como objetivo primordial assegurar de forma eficiente a proteção dos ativos de informação.

A Trust Systems integra o grupo Inoweiser, um ecossistema de empresas cuja missão é utilizar a tecnologia e a inovação para atender às necessidades dos seus clientes. Este grupo tem mais de 19 anos de experiência consolidada, desenvolvendo projetos em mais de 20 países diferentes.

Como mencionado, a Trust tem a missão de ajudar as empresas a proteger os seus dados e a sua privacidade. Face a este propósito, a empresa concebeu nos últimos anos um produto de anonimização de dados que permite aos utilizadores alterar dados e torná-los utilizáveis por terceiros de modo a que não possam ser associados a nenhuma pessoa. O produto tem diferentes tipos de anonimização, oferecendo uma solução descomplicada de proteger informação confidencial.

Com base neste histórico e na área principal de conhecimento, surge o tema Privacidade Diferencial no contexto da Trust Systems. Sendo uma abordagem inovadora e eficaz, o número de organizações que aplicam esta técnica para proteger dados confidenciais tem vindo a crescer cada vez mais. Reconhecendo a importância de se alinhar a esta tendência, a empresa demonstra agora interesse em explorar esta nova abordagem de proteção de dados com objetivo de adquirir uma base de conhecimento que procure reforçar e melhorar a qualidade dos seus serviços e produtos.

Assim, fruto deste interesse e curiosidade neste tema, surge esta proposta de tese. O principal objetivo da mesma passará por estudar e implementar uma ferramenta de Privacidade Diferencial com o intuito de proteger informações consideradas sensíveis durante a análise de grandes quantidades de dados. Esta solução envolverá o estudo de conceitos relevantes e potenciais ferramentas auxiliares a utilizar, bem como uma análise cuidada da estratégia de implementação da solução.

Na formulação do plano do projeto, é fundamental considerar o elemento essencial na execução dos nossos objetivos: o contexto da empresa. A conceção inicial da mesma deve estar alinhada com as suas necessidades, assegurando que a sua implementação esteja em conformidade com esse foco. Sendo uma proposta da própria empresa e voltada para a exploração de um novo tema de estudo, é importante que esse alinhamento se mantenha ao longo de todo o desenvolvimento.

1.3 Contribuições

Nesta secção, apresentam-se as principais contribuições deste trabalho, destacando os avanços alcançados tanto do ponto de vista teórico quanto prático. As contribuições fundamentais são:

- Investigação detalhada e completa da técnica de Privacidade Diferencial, analisando todo o seu funcionamento, os mecanismos disponíveis e as diferentes abordagens de implementação, com o objetivo de identificar qual a melhor forma de a integrar na Trust Systems.

- Criação e implementação de um *middleware* personalizado, incluindo o desenvolvimento de um *plugin* para Traefik elaborado a partir do zero, capaz de intercetar, processar e modificar respostas completas de uma API, utilizando a linguagem de programação GO.

- Formulação e desenvolvimento de um módulo ou microserviço dedicado à aplicação de algoritmos de Privacidade Diferencial, assegurando a incorporação de diferentes mecanismos e a gestão eficiente de parâmetros críticos como, por exemplo, o orçamento de privacidade.

1.4 Estrutura do Documento

O presente documento está estruturado em diferentes capítulos, cada um dedicado a um conjunto específico de tópicos. A seguir, apresenta-se a organização geral de todo o projeto:

- **Capítulo 2:** Apresenta os conceitos teóricos fundamentais, incluindo a origem do tema, as definições formais, os principais mecanismos, bem como as limitações dessa técnica.

- **Capítulo 3:** Focado no trabalho relacionado, analisando e discutindo bibliotecas, frameworks e alguns casos reais de aplicação da Privacidade Diferencial em empresas de referência, com o objetivo de selecionar a abordagem mais adequada para integrar no presente projeto.

- **Capítulo 4:** Descreve a fase de implementação, abrangendo o levantamento de requisitos, a seleção das tecnologias utilizadas e uma descrição detalhada de todo o desenvolvimento da solução, incluindo o *middleware* personalizado e o Módulo de Privacidade Diferencial.

- **Capítulo 5:** Destinado à apresentação dos testes realizados sobre a solução desenvolvida, com análise detalhada dos resultados experimentais e discussão crítica dos mesmos.

- **Capítulo 6:** Sintetiza o trabalho realizado ao longo do projeto, apontando algumas das limitações da solução desenvolvida e indicando sugestões de direções para trabalhos futuros.

Capítulo 2

Privacidade Diferencial

2.1 História e Origem

Em 1977, Tore Dalenius, um estatístico sueco com contribuições significativas na área da privacidade de dados, realizou um trabalho no qual apresentou e desenvolveu de forma significativa uma teoria relacionada com bases de dados. Ele defendia que "nada sobre um indivíduo deve ser aprendido a partir de uma base de dados que não possa ser aprendido sem acesso a essa mesma base de dados" [8]. Esta sua suposição dava a entender que apenas teríamos uma base de dados com garantias de privacidade e confidencialidade se, mesmo tendo acesso a ela, não fosse possível aprender informações sobre um indivíduo que não pudessem ser obtidas a partir de fontes externas ou conhecimento já existente. Por outras palavras, a base de dados não deverá permitir inferir nada de novo sobre um indivíduo em relação ao que já seria possível saber sem o acesso à mesma.

Embora muito difícil de alcançar, essa suposição revolucionou a área da privacidade, incentivando a criação de técnicas de anonimização e proteção de dados. Assim, o estudo de Dalenius abriu portas para investigações sobre soluções que procurassem atingir a sua visão inicial.

Quase três décadas depois, em 2006, Cynthia Dwork [12], juntamente com alguns colegas, investigou formalmente a possibilidade de atingir o objetivo de Dalenius. O estudo revelou que essa ideia é, na verdade, impossível de alcançar. Esta conclusão resulta da existência de informações auxiliares que podem estar disponíveis ao indivíduo, ou seja, dados externos à base de dados que, quando combinados com informações obtidas dela, podem expor informações sensíveis.

Este resultado foi revolucionário, pois demonstrou que o objetivo absoluto de Dalenius não só era inalcançável, mas também que a própria privacidade de um indivíduo poderia ser comprometida, mesmo que este não estivesse presente na base de dados. Esta descoberta mudou a área da privacidade de dados, indicando que uma nova abordagem era necessária.

Face a esse cenário, Dwork e colegas propuseram a Privacidade Diferencial, uma solução mais realista para proteger a privacidade de indivíduos. Ao invés de tentar garantir que ninguém consiga aprender nada sobre um indivíduo, como sugerido por Dalenius, a Privacidade Diferencial propõe adicionar ruído aos dados de forma controlada, equilibrando a sua utilidade e privacidade, de modo a conseguirmos realizar uma análise estatisticamente precisa das informações disponíveis.

2.2 Conceito Inicial

A Privacidade Diferencial é uma técnica matemática rigorosa que assegura a proteção da privacidade individual ao serem partilhadas informações referentes a um determinado grupo. Este método baseia-se na introdução de ruído aleatório, isto é, informações deliberadamente imprecisas nos dados, sendo a magnitude desse mesmo ruído calculada cuidadosamente de modo a que seja suficientemente elevada a ponto de impedir a identificação de informações individuais, mas pequena o suficiente para preservar a utilidade estatística dos dados para futuras análises[18].

Dessa forma, esta abordagem garante que, ao analisar os resultados, qualquer pessoa poderá chegar às mesmas conclusões sobre os dados, de ter conhecimento sobre a presença ou ausência das informações de um indivíduo específico na análise, salvaguardando assim o seu principal objetivo: manter a confidencialidade e privacidade de cada indivíduo preservada. [48].

Por exemplo, imaginemos uma base de dados com milhares de pacientes de um hospital, com atributos como idade, localidade e doença. Embora esses dados possam parecer anónimos à primeira vista, a combinação de idade e localidade pode ser suficiente para identificar um certo paciente. Imaginemos uma pequena localidade de 5.000 habitantes, incluindo 10 pacientes com mais de 60 anos. Se alguém divulgasse estatísticas exatas sobre essa faixa etária, essas informações, combinadas com outras fontes externas, poderiam permitir identificar um paciente específico.

É neste contexto que se evidencia a importância da Privacidade Diferencial. Ao adicionar alguma aleatoriedade em determinados dados, como, por exemplo, estatísticas sobre as idades dos pacientes, a técnica garante que um analista não consiga deduzir informações sobre um indivíduo específico. No entanto, continua a permitir a extração de informações úteis sobre a prevalência de doenças numa determinada faixa etária, preservando a privacidade dos pacientes.

2.3 Definição Formal

Compreendido o conceito geral de Privacidade Diferencial, avançamos para a sua definição formal.

Esta técnica é formalmente definida utilizando um parâmetro ϵ, um número real positivo conhecido como orçamento de privacidade, que determina o nível de privacidade a adicionar aos dados. Essa formalização envolve ainda um mecanismo aleatório, isto é, um algoritmo que, a partir de um conjunto de dados que recebe, gera como saída uma resposta a possíveis operações como somas, médias ou contagens, dependendo do tipo de análise que está a ser efetuada.

Segundo a investigadora Cynthia Dwork [12, 25], dado um mecanismo aleatório M, esse mesmo mecanismo garante ϵ - Privacidade Diferencial se para todos os conjuntos de dados vizinhos, $D1$ e $D2$, isto é, conjuntos que diferem no máximo num único registo (um conjunto possui um registo a mais ou a menos em relação ao outro), e para todo o subconjunto S contido no intervalo de todos os resultados possíveis provenientes de M, temos:

$$P[M(D1) \in S] \leq e^{\epsilon}\, P[M(D2) \in S]$$

Onde, $P[M(D1) \in S]$ é a probabilidade do mecanismo M gerar um resultado em S quando aplicado ao conjunto de dados $D1$, e $P[M(D1) \in S]$ é a probabilidade para $D2$. A desigualdade presente na expressão garante que a diferença entre as probabilidades de resultados seja limitada pela razão e^{ϵ}. Quando ϵ é pequeno, e^{ϵ} estará próximo de 1, logo ambas as probabilidades serão muito próximas, ou praticamente semelhantes. Isto garante que a presença ou ausência de um certo registo não afete muito o resultado, garantindo maior privacidade. Pelo contrário, se ϵ for maior, as probabilidades podem diferir mais, o que implica menor privacidade. [37]

O parâmetro ϵ [21] desempenha um papel importante, já que funciona como um regulador que permite controlar o equilíbrio entre a precisão e a privacidade dos dados. Quanto menor o orçamento de privacidade, maior será a privacidade, mas menor será a precisão. Pelo contrário, quanto maior o ϵ, menor será a privacidade e, como consequência, a precisão dos dados será maior.

Para além da definição já descrita, podemos introduzir uma outra que simplesmente adiciona um novo parâmetro para a nossa análise. Falamos agora da (ϵ, δ) - Privacidade Diferencial, uma nova ideia que permite uma pequena violação da privacidade, isto é, uma pequena probabilidade do mecanismo de Privacidade Diferencial falhar e não proteger completamente a privacidade de um determinado indivíduo. A condição desta privacidade apresenta-se da seguinte forma:

$$P[M(D1) \in S] \leq e^{\epsilon}\, P[M(D2) \in S] + \delta$$

Tal como é possível verificar, mantém-se tudo como anteriormente, contudo, neste caso, adiciona-se o parâmetro δ, um valor muito pequeno, geralmente no intervalo de 1×10^{-3} a 1×10^{-12}, sendo o valor mais comum igual a 1×10^{-5}, que permite tornar esta ideia mais flexível, permitindo a introdução de alguma incerteza na privacidade, embora com um risco aceitável. [3]

Assim, apresentámos os dois conceitos formais desta técnica: Privacidade Diferencial Pura, se $\delta = 0$, garantindo uma forte privacidade dos dados em análise, e Privacidade Diferencial Aproximada, caso $\delta > 0$, permitindo uma pequena margem de risco, valorizando mais a utilidade dos dados para análises estatísticas, mas mantendo na mesma um elevado nível de privacidade.

2.4 Duas Diferentes Abordagens

Tal como descrito inicialmente, a Privacidade Diferencial baseia-se na introdução de aleatoriedade nos dados em análise. O que falta agora entender é que esse processo pode ocorrer de maneiras diferentes, dependendo da configuração adotada. Atualmente, os investigadores reconhecem duas diferentes abordagens de Privacidade Diferencial: Privacidade Diferencial Local e Privacidade Diferencial Global (ilustradas na Figura 2.1), sendo cada uma delas descrita já a seguir [16, 7]:

Privacidade Diferencial Local

Nesta primeira abordagem, cada indivíduo adiciona ruído aos seus próprios dados antes de os enviar, garantindo que ninguém sabe o valor exato dos dados com exceção do próprio indivíduo.

Esta abordagem é útil quando se deseja proteger os dados antes de serem recolhidos pelo *Agregador*, entidade responsável por recolher e agregar os dados dos indivíduos para gerar estatísticas ou análises. Este método é interessante quando não se confia totalmente nessa mesma entidade.

No entanto, a Privacidade Diferencial Local apresenta uma desvantagem: como cada pessoa adiciona sua própria deturpação aos dados, o resultado final pode ficar menos preciso. Desta forma, é necessário recolher uma maior quantidade de dados de modo a compensar essa imprecisão e assegurar resultados mais confiáveis, especialmente em situações onde a precisão é essencial.

Privacidade Diferencial Global

Neste tipo de abordagem, os dados são recolhidos tal e qual como estão, sem ruído e deturpação, e o responsável por armazenar e processar os dados, designado neste caso de *Curador*, adiciona o ruído apenas ao resultado final das consultas efetuadas por um determinado utilizador.

Para esta abordagem funcionar de forma correta, é necessário confiar plenamente no *Curador* de dados, uma vez que esta importante entidade tem acesso completo a todos os dados originais, devendo, por isso, protegê-los a todo o custo. Nestas circunstâncias, a precisão dos resultados obtidos é muito superior quando comparada com a abordagem anterior, uma vez que o ruído só é aplicado no final de todo o processo, apresentando um menor impacto na utilidade dos dados.

Visto pelo lado negativo, toda a responsabilidade está centrada no *Curador* e, como tal, caso ocorra uma falha de segurança, todos os dados ficam completamente expostos. Este é um ponto muito importante, já que essa rutura de privacidade pode comprometer informações sensíveis, prejudicar os indivíduos envolvidos e afetar a confiança em todo o sistema, podendo ainda envolver eventuais consequências legais e financeiras para a entidade responsável pelo mesmo.

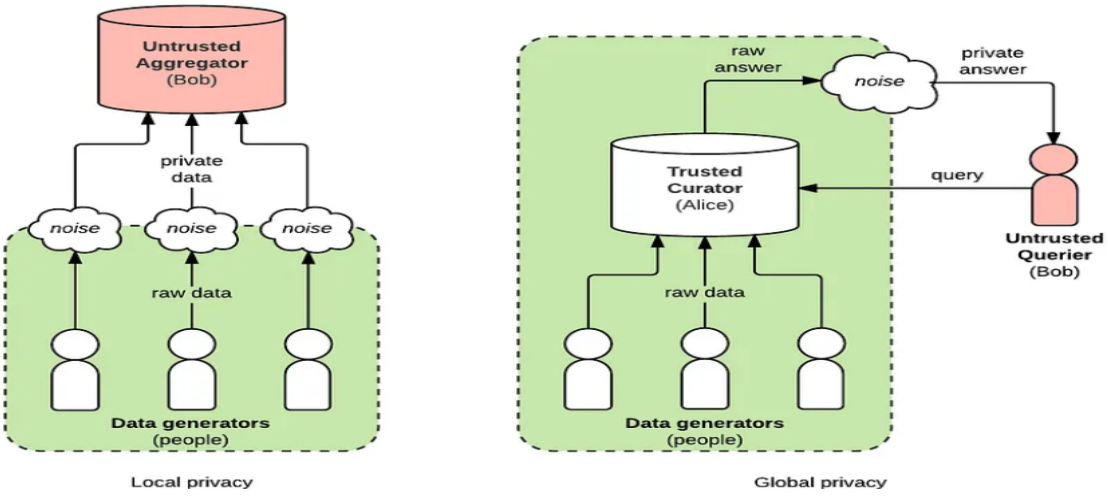

Figura 2.1: Privacidade Diferencial Local vs Global

De uma forma mais resumida, a Privacidade Diferencial Local é usada quando há uma grande preocupação com a privacidade do utilizador e não se pode confiar totalmente no *Agregador* de dados. Já a Privacidade Diferencial Global é aplicada quando o *Curador* dos dados é confiável, permitindo um processamento mais preciso das informações, mas ainda mantendo um certo nível de privacidade. Esse modelo oferece melhores resultados quando os dados são processados por uma entidade de confiança, garantindo precisão sem comprometer a privacidade do indivíduo.

2.5 Funcionamento Geral

A Privacidade Diferencial tem como principal objetivo garantir a proteção de informações individuais, enquanto permite análises de grandes quantidades de dados. O seu procedimento baseia-se na aplicação de certos mecanismos que, ao inserirem informação aleatória e imprecisa, asseguram a possibilidade de análise de informação coletiva, nunca comprometendo dados individuais. Assim, toda esta técnica envolve etapas complexas que requerem especial atenção.

O processo inicia-se com o utilizador a realizar uma consulta ao sistema que recolhe toda a informação. Essa consulta pode ser simples ou complexa, cabendo ao utilizador saber qual a informação que pretende obter. Em vez de ser enviada diretamente para a base de dados, ela passa primeiro por um software intermédio, conhecido como Guarda de Privacidade. Esse intermediário tem a função de avaliar o impacto na privacidade dessa consulta, aplicando um mecanismo com um certo algoritmo que mede o risco da resposta em comprometer informações individuais.

Após esta avaliação, a consulta é enviada à base de dados, sendo retornada uma resposta limpa, isto é, sem qualquer alteração ao valor correto. Posteriormente, o Guarda aplica, através de um certo mecanismo selecionado, uma quantidade apropriada de ruído controlado à resposta, ajustando o nível de distorção conforme o risco identificado inicialmente pelo próprio. Estando agora a resposta já protegida, esta é entregue ao utilizador como resposta final à consulta.

Desta forma, ao longo de todo o processo, o equilíbrio entre utilidade e privacidade dos dados é fundamental. O software intermediário assegura que as respostas sejam suficientemente precisas para análises gerais, mas também imprecisas o suficiente para garantir a proteção da privacidade dos indivíduos. Para isso, o mecanismo implementado ajusta o nível de ruído com base em dois conceitos essenciais: a sensibilidade dos dados e o orçamento de privacidade. Estes são dois elementos fundamentais que se relacionam ao longo de todo este processo, trabalhando em conjunto para definirem qual a deturpação que será aplicada ao resultado da consulta do utilizador. [37, 34]

Figura 2.2: Funcionamento Privacidade Diferencial

2.6 Orçamento de Privacidade e Sensibilidade

Como discutido anteriormente, o orçamento de privacidade é responsável por quantificar o nível de proteção fornecido por um determinado mecanismo de privacidade, isto é, controla a quantidade de informação que pode ser retirada de um certo conjunto de dados sem que a privacidade de um indivíduo específico seja comprometida. Quanto menor for o valor de ϵ, maior será a proteção dos dados, mas menor será a sua precisão. Pelo contrário, quanto maior o valor de ϵ, menor será a proteção dos mesmos, mas maior será a precisão da informação obtida.

Assim, estamos perante o maior desafio da implementação da Privacidade Diferencial, a escolha do orçamento de privacidade. Esta seleção tem de ser muito bem ponderada, uma vez que é este o valor responsável por controlar todo este equilíbrio que acabámos de descrever.

Quanto à escolha deste valor, o orçamento de privacidade deve ser selecionado com base numa série de considerações técnicas, tendo em conta o equilíbrio entre a utilidade e a privacidade dos dados. Embora não existindo um consenso sobre o valor ideal para o orçamento de privacidade, ele é geralmente definido com valores entre 0.01 e 1, dependendo das necessidades e contexto da implementação em questão. A recomendação de alguns investigadores é que o valor deve ser pequeno, mas não tão pequeno de modo a tornar a análise inútil. Um valor de $\epsilon = 0.1$ é considerado, pelos mesmos, como um valor razoável para muitas das implementações da técnica.

Este orçamento funciona como um recurso limitado, na medida em que é consumido à medida que cada consulta é realizada. Cada uma dessas consultas utiliza parte desse orçamento, sendo a quantidade consumida calculada com base no mecanismo de privacidade adotado e na sensibilidade dos dados envolvidos. Por exemplo, se o orçamento inicial for de 0,1 e duas consultas consumirem 0,01 e 0,02, o valor restante será 0,07 para as consultas seguintes. Caso o orçamento atinja o seu limite, isto é, caso a soma de todos os valores de ϵ ultrapasse o orçamento total, deve-se interromper de imediato a realização de novas consultas, garantindo que a privacidade dos dados não seja posta em causa. Para além de restringir os utilizadores de prosseguirem com novas e futuras análises, o sistema pode, em certos e raros casos de implementação, notificar os responsáveis pela mesma de modo a sugerir uma possível alteração do valor de ϵ. [48, 13].

Entendido o orçamento de privacidade, podemos agora introduzir um novo conceito: a sensibilidade dos dados, uma das definições mais cruciais da Privacidade Diferencial. Esse parâmetro mede o quanto o resultado de uma consulta pode variar quando um único registo é alterado ou removido da base de dados, determinando a quantidade de ruído a ser adicionado à resposta.

Por exemplo, numa consulta que calcula o número de pacientes com uma determinada doença, a remoção ou adição de um único paciente apenas consegue alterar o resultado no máximo numa unidade. Como tal, neste caso concreto, o valor da sensibilidade vai ser sempre igual a um.

De modo geral, conseguimos rapidamente entender que a quantidade de deturpação necessária para garantir a privacidade dos dados é diretamente proporcional à sensibilidade da consulta. Quanto maior a sensibilidade, maior o ruído necessário para preservar a privacidade. Consequentemente, para manter um nível fixo de privacidade, valores menores de ϵ serão consumidos

em cada consulta, de modo a equilibrar a proteção dos dados com a utilidade do resultado [23].

Assim, sensibilidade e orçamento de privacidade são dois termos muito importantes neste contexto, trabalhando ambos em conjunto ao longo de todo este processo. Juntos, eles asseguram que as informações divulgadas nas respostas sejam suficientemente úteis para o utilizador, ao mesmo tempo que protegem a identidade de todos os indivíduos envolvidos.

2.7 Principais Propriedades

A Privacidade Diferencial possui propriedades importantes [9, 14] que garantem o bom funcionamento da técnica descrita nas secções anteriores. Essas são fundamentais, pois garantem que, independentemente da forma como as consultas são efetuadas, a privacidade de todos os indivíduos é mantida pelo orçamento de privacidade estabelecido. Apresentam-se, a seguir, quatro propriedades que desempenham um papel crucial na garantia da eficácia desta técnica:

1. **Composição Sequencial:** Quando temos vários mecanismos diferentes a serem aplicados de forma sequencial, ou seja, um após o outro, a privacidade do conjunto de mecanismos vai ser igual à soma dos níveis individuais de privacidade de cada um deles. Por outras palavras, se forem efetuadas várias consultas, cada uma com um determinado valor de ϵ atribuído, o valor total de privacidade nestes casos é dado pela soma dos ϵ de cada consulta.

2. **Composição Paralela:** Quando os mecanismos são aplicados em subconjuntos, isto é, em partes diferentes e mais pequenas do conjunto de dados, o valor da privacidade é determinado pela partição que tiver o maior valor de ϵ. Assim, se tivermos dois conjuntos diferentes e forem aplicados diferentes mecanismos de privacidade em cada um deles, a privacidade do conjunto total será igual à privacidade do mecanismo com maior risco.

3. **Pós-Processamento:** Segundo esta propriedade, qualquer função definida usando os resultados previamente obtidos a partir de um mecanismo diferencialmente privado, continuará a ser diferencialmente privado. Como tal, mesmo após a aplicação de novas funções sobre esses dados, o nível de privacidade não será comprometido, salvaguardando a sua preservação.

4. **Privacidade de Grupos:** Propriedade que permite garantir não apenas a privacidade de indivíduos isolados, mas também a proteção de grupos inteiros que possuam alguma relação entre si, como famílias ou membros de uma mesma comunidade. A técnica assegura que a presença de um determinado grupo com relações entre si não seja identificada num conjunto de dados, preservando informações sensíveis sobre todo esse conjunto de indivíduos.

2.8 Mecanismos de Privacidade Diferencial

Tal como vimos nas secções anteriores, quando falamos de mecanismo, estamos a referir-nos a um algoritmo matemático cuja principal função é introduzir ruído aleatório aos dados e às respostas a consultas realizadas. Sendo esta uma etapa chave deste processo, vamos agora abordar os diferentes tipos de mecanismos existentes e as situações em que cada um deles deve ser utilizado.

Mecanismo de Laplace

O mecanismo de Laplace [14, 36] é um dos métodos mais simples e fundamentais da Privacidade Diferencial, sendo amplamente utilizado em consultas numéricas. Ele funciona adicionando ruído, que segue uma distribuição Laplace, ao resultado da consulta, sendo a intensidade do mesmo ajustada de acordo com a sensibilidade de cada consulta em questão. Este mecanismo apresentado satisfaz ϵ - Privacidade Diferencial e pode ser formalmente descrito da seguinte forma:

$$F(x) = f(x) + Lap\left(\frac{s}{\epsilon}\right)$$

Onde $f(x)$ é o resultado da consulta original, $F(x)$ é o resultado da consulta após a adição de ruído, s é a sensibilidade da consulta e ϵ é o orçamento de privacidade.

Mecanismo Gaussiano

O Mecanismo Gaussiano [14, 36] é das alternativas ao Mecanismo de Laplace, sendo também utilizado em consultas com dados numéricos. Ele funciona adicionando ruído, proveniente de uma distribuição Gaussiana, ao resultado de uma consulta. Ao contrário do Mecanismo de Laplace, que garante Privacidade Diferencial Pura, ou seja, $(\epsilon, 0)$ - Privacidade Diferencial, o Mecanismo Gaussiano satisfaz Privacidade Diferencial Aproximada, logo, (ϵ, δ) - Privacidade Diferencial.

Neste método, a quantidade de ruído a ser adicionada depende não só dos dois parâmetros já conhecidos do Mecanismo Laplace (orçamento de privacidade e sensibilidade), mas também do parâmetro δ, uma variável que permite uma pequena probabilidade de violação de privacidade. Esse novo parâmetro torna o mecanismo mais flexível, permitindo cenários em que uma leve falha na privacidade é aceitável. Dessa forma, podemos descrevê-lo formalmente da seguinte maneira:

$$F(x) = f(x) + \mathcal{N}(\sigma^2) \ , \ \text{onde} \ \ \sigma^2 = \frac{2s^2 \log\left(\frac{1.25}{\delta}\right)}{\epsilon^2}$$

Onde $f(x)$ é o resultado da consulta original, $F(x)$ é o resultado da consulta após adição de ruído, s é a sensibilidade da consulta, ϵ é o orçamento de privacidade e $\mathcal{N}(\sigma^2)$ representa o ruído gaussiano com variância σ^2.

O Mecanismo Gaussiano é mais adequado para cenários que exigem alta privacidade, ou seja, quando o valor de ϵ é pequeno (geralmente menor ou igual a 1). Em contrapartida, o Mecanismo de Laplace é mais versátil, podendo ser aplicado em cenários de privacidade variados e mais simples.

Mecanismo Exponencial

O Mecanismo Exponencial [36] é um método que permite selecionar o "melhor" elemento de um conjunto de opções enquanto preserva a privacidade dos dados. O seu funcionamento é estruturado em três diferentes etapas: primeiro, define-se um conjunto de opções possíveis, representado por $\mathcal{R}$; em seguida, especifica-se uma função de pontuação $u(x, r)$, onde x são os dados analisados e r pertence a $\mathcal{R}$, sendo essa função responsável por atribuir uma certa pontuação a cada opção; finalmente, o mecanismo seleciona um desses elementos com uma probabilidade proporcional a:

$$\exp\left(\frac{\epsilon \cdot u(x, r)}{2\Delta u}\right)$$

Onde Δu é a sensibilidade da função de pontuação, isto é, a maior variação que a pontuação de um elemento pode sofrer ao ser alterado um único dado no conjunto.

A função exponencial demonstrada é utilizada para transformar a pontuação numa probabilidade. A exponencial aumenta rapidamente à medida que a pontuação também aumenta, o que significa que os elementos com uma pontuação mais alta têm uma probabilidade muito maior de serem selecionados. Tudo isto é uma forma de garantir que este mecanismo prioriza as opções mais "relevantes", mas ao mesmo tempo preserva a privacidade dos dados considerados sensíveis.

Assim, o resultado deste mecanismo é sempre um elemento desse conjunto $\mathcal{R}$, tornando-o um método ideal quando temos consultas com dados categóricos ou discretos, ou seja, em situações onde a resposta precisa de ser uma escolha específica de entre um conjunto finito de opções.

Os três mecanismos descritos [2] representam técnicas matemáticas complexas e robustas, evidenciando o caráter matemático que está por dentro desta técnica de anonimização que é a Privacidade Diferencial. Embora apresentados de forma teórica nesta secção, todos estes mecanismos serão posteriormente utilizados e demonstrados na prática durante a fase de implementação do projeto, permitindo avaliar seu desempenho e aplicabilidade em contextos reais.

O objetivo desta secção foi fornecer uma visão geral dos mecanismos, destacando os critérios que levam à escolha de cada e possibilitando uma compreensão da sua utilidade e funcionalidade. De modo a sintetizar todas estas informações, os mecanismos são resumidos na Tabela 2.1.

Mecanismo	Tipo de Consulta	Privacidade	Ruído Adicionado
Laplace	Numérica	$(\epsilon, 0)$	$Lap\left(\frac{s}{\epsilon}\right)$
Gaussiano	Numérica	(ϵ, δ)	$\mathcal{N}\left(\frac{2s^2 \log\left(\frac{1.25}{\delta}\right)}{\epsilon^2}\right)$
Exponencial	Categórica / Discreta	$(\epsilon, 0)$	Ruído Probabilidade

Tabela 2.1: Mecanismos de Privacidade Diferencial

2.9 Desafios e Limitações

A Privacidade Diferencial, com base na análise já realizada, proporciona uma abordagem mais eficaz e confiável para a proteção de dados, quando comparada com outras técnicas já existentes como, por exemplo, k-anonimização [45] e pseudonimização [10]. O facto de apresentar uma garantia matemática devidamente fundamentada garante que as chances de identificar informações sobre um indivíduo específico sejam matematicamente limitadas, independentemente da existência de informações externas que possam, de certa forma, ser usadas para inferir dados pessoais.

Contudo, é importante destacar que a probabilidade de tal identificação depende de fatores fundamentais como o orçamento de privacidade e a sensibilidade dos dados. Para além disso, esta é uma técnica única na medida em que permite quantificar a possível perda de privacidade através do orçamento de privacidade e da sua característica de composição. Pelo contrário, os restantes métodos tradicionais existentes geralmente baseiam-se em supressões ou generalizações de dados que podem levar à remoção de informações identificáveis ou à redução da precisão dos dados.

Para além disso, alguns estudos descritos em [48] têm comprovado que essas técnicas são suscetíveis a possíveis ataques de reidentificação quando combinadas com outras fontes públicas externas, tornando grande parte dos seus dados expostos a possíveis violações de privacidade.

Apesar de tudo isto, o tema principal deste projeto também apresenta as suas limitações e desafios que continuam a ser investigados e analisados [18] . Um dos principais obstáculos prende-se com a definição do parâmetro ϵ. Com todos estes anos de estudo e investigação relacionados com o tema, o valor ideal a adotar para esta variável ainda é uma incógnita que carece de padrões claros para a sua definição. Sendo um parâmetro essencial e fulcral ao longo de todo este processo,

é muito importante a procura desse valor em cada contexto de implementação desta técnica.

A possibilidade de realização de consultas repetidas por um indivíduo também pode ser considerada um obstáculo. Isso ocorre porque, caso o orçamento global estabelecido seja elevado, o indivíduo pode efetuar inúmeras consultas repetidas e, a partir desses dados, inferir o seu valor real. Assim, é fundamental que sejam incluídas estratégias para minimizar essa vulnerabilidade.

Quanto à complexidade computacional, esta pode ser vista como uma limitação na medida em que muitas aplicações desta técnica exigem elevados recursos computacionais, o que a torna não apenas difícil, mas também custosa e, em alguns casos, inviável em termos de desempenho.

Outro ponto relevante é o facto desta abordagem funcionar melhor com grandes conjuntos de dados, isto é, com milhões de registos. Pelo contrário, quando aplicada a conjuntos de dados muito pequenos, torna-se desafiador proteger as informações sem comprometer sua utilidade estatística.

Por fim, é importante mencionar que, embora a Privacidade Diferencial proteja contra ataques de privacidade, como reidentificação (processo que consiste na identificação de indivíduos em dados anonimizados quando cruzados com fontes externas), ela não oferece proteção contra ataques de segurança, como tentativas de acesso não autorizado. Assim, é importante garantir que o ambiente onde esta técnica é implementada esteja devidamente protegido de modo a evitar violações de segurança que possam comprometer a integridade e a confidencialidade de todos os dados.

Capítulo 3

Trabalho Relacionado

Neste capítulo serão abordadas investigações previamente realizadas e consideradas relevantes para o contexto em questão. Serão abordadas as principais ferramentas já utilizadas na área e será feita referência a duas empresas de renome, sendo descrita a forma como utilizam esta técnica.

3.1 Bibliotecas e Frameworks

A Privacidade Diferencial, tal como já descrito, tem ganho destaque nos últimos anos, tendo sido desenvolvidas, ao longo dos anos, inúmeras ferramentas, bibliotecas e frameworks que permitem aplicar esta técnica em diferentes linguagens de programação, plataformas e contextos.

No trabalho de Shiliang Zhang et al. [49], foi proposta uma estrutura de avaliação de código aberto para soluções de proteção de privacidade de dados. Nesse estudo, o investigador, juntamente com outros especialistas, apresenta uma avaliação base de algumas das ferramentas existentes relacionadas com esta técnica. Gonzalo M. Garrido et al. [19], por sua vez, destaca na sua investigação não apenas os requisitos essenciais, mas também as limitações das ferramentas atualmente disponíveis, tendo como objetivo orientar potenciais utilizadores quanto ao uso adequado e eficaz dessas soluções. Além disso, Anshu Singh et al. [46], num dos seus artigos relativos à Privacidade Diferencial, oferece uma análise crítica e detalhada da maioria das ferramentas emergentes, propondo uma avaliação comparativa das mesmas. Esta abordagem comparativa permite, assim, destacar diferenças em termos de linguagens suportadas, mecanismos e algoritmos oferecidos.

Com base em todas estas investigações, é possível identificar as ferramentas mais relevantes para a aplicação desta estratégia em diferentes cenários. A seguir, serão apresentadas e descritas algumas, levando em consideração todas as suas características e funcionalidades:

A. Google Differential Privacy: A biblioteca open-source disponibilizada pela Google [20] é uma versão da sua biblioteca de Privacidade Diferencial utilizada em alguns dos seus inúmeros produtos. Está disponível em três diferentes linguagens de programação (Java, C++ e Go), apresentando um nível introdutório de conhecimento ideal para aqueles que estão a iniciar no tema.

B. OpenDP: Biblioteca [38] que apresenta uma coleção de módulos de algoritmos estatísticos que permitem a aplicação da técnica de Privacidade Diferencial. Uma das suas principais características é o facto de apresentar uma funcionalidade que calcula parâmetros como a sensibilidade de uma consulta. Faz parte do OpenDP Project, um projeto comunitário para criar ferramentas de análise de dados, tendo sido desenvolvida em Rust, oferecendo bindings para Python e R.

C. Diffprivlib: Biblioteca desenvolvida pela IBM (International Business Machines) [27] que permite a aplicação de Privacidade Diferencial em diferentes tarefas de Machine Learning como classificação, regressão, agrupamento, redução de dimensionalidade e regularização de dados. É uma biblioteca open-source com suporte para a linguagem Python (versões 3.8 a 3.12) que permite, para além da realização desses processos de Machine Learning, executar várias consultas de agregação diferencialmente privadas. Um dos seus pontos de destaque é o facto de possuir um contador que permite controlar o orçamento de privacidade usando técnicas de composição.

D. Tensorflow Privacy: A biblioteca TensorFlow Privacy [44] é uma biblioteca open-source criada por equipas da Google Research. É desenvolvida em Python e inclui implementações de otimizadores TensorFlow utilizados para treinar modelos de Machine Learning com Privacidade Diferencial. Para além disso, a biblioteca em questão disponibiliza um conjunto diversificado de tutoriais e ferramentas de análise, permitindo uma utilização muito mais eficiente, flexível e eficaz.

E. Opacus: Biblioteca [43] de alto desempenho do Facebook que tem como intuito treinar modelos com Privacidade Diferencial de forma muito eficiente. Foi desenvolvida em colaboração com o Facebook AI Research, a equipa do PyTorch e do OpenMined, uma comunidade open-source dedicada ao desenvolvimento de técnicas de privacidade para Machine Learning e Inteligência Artificial. Toda esta biblioteca, desenvolvida em Python, requer poucas alterações no código do cliente, oferecendo um baixo impacto no desempenho de treino. Mais uma vez, esta biblioteca possibilita o monitoramento em tempo real do valor do orçamento de privacidade a ser utilizado.

F. Tumult Analytics: Desenvolvida pela Tumult Labs [30], uma startup especializada em soluções de Privacidade Diferencial, oferece uma biblioteca em Python construída sobre uma estrutura semelhante à OpenDP. Esta biblioteca permite o cálculo de estatísticas agregadas sobre grandes volumes de dados, disponibilizando uma interface intuitiva e de fácil utilização, suporte a uma ampla gama de funções de agregação, operadores de transformação e mecanismos configuráveis.

G. PipelineDP: Framework [39] desenvolvida pela OpenMined, em colaboração com a Google. Permite ao utilizador executar consultas agregadas diferencialmente privadas em conjuntos de dados usando sistemas de processamento em lote como o Apache Spark e o Beam. Suporta a linguagem de programação Python, contudo, importa salientar que esta ferramenta ainda está numa fase mais experimental, não sendo recomendada a sua utilização em sistemas já em produção.

Como é possível observar, todas as ferramentas descritas fornecem uma gama de capacidades que podem ser utilizadas para atender a diferentes requisitos no campo da Privacidade Diferencial. Foram desenvolvidas para suporte a várias linguagens de programação, vários algoritmos e componentes de privacidade, o que garante enorme flexibilidade para todo o tipo de projetos.

A seleção de uma das ferramentas da lista anterior depende de vários fatores, como áreas de aplicação do projeto, o nível de complexidade envolvido, os objetivos pretendidos e a framework de desenvolvimento. Para além disso, algumas dessas ferramentas são mais aconselhadas para iniciantes porque possuem uma implementação simplificada e uma documentação acessível, enquanto outras são mais complexas e fornecem integrações com sistemas de Machine Learning.

Uma característica comum a todas as ferramentas mencionadas, e que merece total destaque, é o facto de ser open source, o que garante acesso livre e transparente ao código-fonte. Esta é uma característica indispensável neste contexto, uma vez que possibilita que seja possível adaptar e integrar o código na nossa solução, tudo isto dentro dos limites estabelecidos pelas suas licenças.

De modo a permitir uma análise mais detalhada e completa de todos os instrumentos que temos à nossa disposição, foi construída a Tabela 3.1, com o intuito de obtermos uma comparação criteriosa entre algumas das características mais relevantes destes produtos aqui abordados.

Características	Google	OpenDP	Diffprivlib	Tensorflow	Opacus	Tumult Analytics	PipelineDP
Linguagem de Programação	Go, Java, C++	Rust (bindings para Python e R)	Python	Python	Python	Python	Python
Sistema Operativo	Windows	Windows	Windows	Windows	Windows	Linux	Windows
Domínio Principal	Análise Estatística	Análise Estatística	Análise Estatística e Machine Learning	Machine Learning	Machine Learning	Análise Estatística	Análise Estatística
Mecanismos Suportados	Laplace, Gaussiano e Exponencial	Laplace e Gaussiano	Laplace, Gaussiano, Exponencial, Binário, Geométrico, Uniforme	Laplace e Gaussiano	Laplace, Gaussiano, Exponencial	Laplace, Gaussiano, Exponencial	Laplace e Gaussiano
Suporta consultas de agregação: COUNT, MAX, MIN, AVG, VAR, SUM	✓	✓	✓	✗	✗	✓	✓
Consultas permitem cláusulas WHERE, GROUP BY e JOIN	✗	✗	✗	✗	✗	✓	✗
Permite acompanhar os gastos do orçamento de privacidade	✓	✓	✓	✓	✓	✓	✓
Bloqueia consultas ao ser esgotado o orçamento de privacidade	✗	✓	✓	✗	✗	✓	✗

Tabela 3.1: Comparação entre diferentes ferramentas de Privacidade Diferencial

A análise da Tabela 3.1 permite tirar conclusões importantes sobre as ferramentas existentes:

- A maioria das bibliotecas é implementada em Python, linguagem utilizada em *Data Science* e *Machine Learning*. No entanto, existem opções desenvolvidas em outras linguagens, como Go, Java, C++ e Rust. Praticamente todas as ferramentas analisadas são compatíveis com o sistema operativo Windows, exceto uma que funciona em Linux, a *Tumult Analytics*.

- O domínio central dessas ferramentas concentra-se principalmente em *Machine Learning* e Análise Estatística, sendo que cada biblioteca oferece suporte para cada uma dessas áreas, com exceção da opção *Diffprivlib*, que suporta ambos os domínios mencionados.

- A maioria das ferramentas oferece os mecanismos principais e essenciais para a proteção de dados numéricos, como o Mecanismo de Laplace e o Mecanismo Gaussiano. Contudo, algumas bibliotecas como *Diffprivlib*, *Opacus* e *Tumult Analytics* oferecem implementação para o Mecanismo Exponencial, fundamental para a proteção de dados categóricos.

- A capacidade de executar consultas de agregação, como COUNT, SUM e AVG, é outra funcionalidade muito relevante. Ferramentas como *Google DP*, *OpenDP*, *Diffprivlib*, *Tumult Analytics* e *PipelineDP* suportam essas operações, enquanto bibliotecas voltadas para *Machine Learning*, como *TensorFlow Privacy* e *Opacus*, não oferecem esse suporte.

- No que ao orçamento de privacidade diz respeito, todas as ferramentas analisadas permitem a gestão eficaz e acompanhamento desse parâmetro crítico. No entanto, a capacidade de bloquear certas consultas quando o orçamento é excedido difere entre as bibliotecas.

3.2 Aplicações Reais

Com o crescimento da Privacidade Diferencial, esta tem sido cada vez mais adotada por grandes empresas que, através das suas funcionalidades, protegem a privacidade dos utilizadores enquanto recolhem informações valiosas para melhorar os seus serviços. De seguida, serão mencionados dois exemplos dessas empresas, bem como a forma como estas atuam perante esta recente técnica.

3.2.1 Apple

A Apple utiliza Privacidade Diferencial de modo a obter informações sobre a forma como os utilizadores utilizam os seus serviços. A empresa [28] desenvolveu um sistema complexo que permite, através da aplicação da técnica de Privacidade Diferencial Local, obter dados aleatórios dos seus utilizadores de maneira a que o seu servidor nunca veja ou receba os dados reais.

Assim, os dados são privatizados ainda no dispositivo do indivíduo, sendo esses posteriormente enviados para o servidor através de um canal cifrado, sem qualquer informação relativa ao dispositivo em questão. Essa transmissão ocorre uma vez por dia, sendo imediatamente descartados todos os endereços de IP, datas e horas em que os mesmos acabaram por ser capturados.

Essas informações são devidamente processadas, possibilitando o cálculo de estatísticas que permitem à empresa obter dados relevantes dos utilizadores. A Apple atribui um determinado orçamento de privacidade a cada atividade possível de ser realizada pelos seus utilizadores, recolhendo diariamente um número limitado de registos privatizados, com base nesse parâmetro.

Os dados ficam temporariamente armazenados no dispositivo, sendo depois enviada apenas uma amostra aleatória dos mesmos ao servidor. Esta técnica é usada desde 2016 com a introdução do iOS 10, estando presente em certas funcionalidades como sugestões do QuickType e sugestões de emojis. De realçar que só é possível realizar todo este processo com os utilizadores que, nas definições do seu próprio dispositivo, aceitem a partilha de dados protegidos para análise.

3.2.2 Google

Em 2014, a Google introduziu Privacidade Diferencial no seu navegador principal, o Google Chrome. Esta técnica foi aplicada através da implementação do mecanismo RAPPOR (*Randomized Aggregatable Privacy-Preserving Ordinal Response*) [15] , um método que permite recolher estatísticas probabilísticas relacionadas com a forma como os utilizadores utilizam este navegador.

Este sistema permite ao navegador converter sequências de caracteres, como um URL, em sequências curtas de bits usando uma função *Hash* que adiciona imediatamente ruído aos dados. Assim, através da Privacidade Diferencial Local, a Google consegue recolher milhares dessas sequências privadas, conseguindo saber quais são as mais frequentes, sem nunca ser possível associar qualquer uma delas a um indivíduo específico, mantendo um elevado nível de segurança.

Todas essas sequências frequentes são posteriormente comparadas com um conjunto de *hashes* de URLs já calculados anteriormente pela Google, permitindo que a empresa consiga estimar quais as rotinas mais comuns entre os utilizadores, como por exemplo, os sites mais visitados ou até mesmo a taxa de cliques num qualquer diálogo com o seu assistente virtual, o Google Assistant.

De mencionar que a empresa disponibilizou publicamente em código aberto a sua biblioteca completa de Privacidade Diferencial (mencionada na Secção 3.1), permitindo que outros desenvolvedores interessados possam facilmente integrar esta abordagem nos seus próprios projetos.

Capítulo 4

Implementação

Neste capítulo estará descrita de forma detalhada toda a implementação da solução, salientando as decisões tomadas nas diferentes fases desta etapa de desenvolvimento. Serão apresentadas as principais características que a nossa implementação possui, todas as tecnologias e ferramentas selecionadas, a descrição da arquitetura da solução e uma explicação bem pormenorizada dos componentes principais e da função desempenhada por cada um destes no contexto em questão.

4.1 Levantamento de Requisitos

A etapa de levantamento de requisitos marca o ponto de partida deste projeto, desempenhando um papel fundamental na definição da solução a desenvolver. Esta fase assume uma particular importância na medida em que permite compreender quais as necessidades da empresa, alinhar expectativas entre as partes envolvidas e consolidar os objetivos da solução a implementar.

No nosso caso em específico, esta etapa não se limitou apenas à simples recolha de detalhes técnicos do produto final. Embora a empresa tenha demonstrado interesse em explorar a técnica de Privacidade Diferencial, por estar diretamente inserida na sua área de atuação, esta ainda não possuía uma solução concreta idealizada, uma vez que o conhecimento sobre o tema era, até então, bastante superficial. Face a este cenário, foi necessária uma abordagem inicial autónoma e investigativa, com o intuito de compreender tudo o que envolvia o tema a ser explorado.

Assim, foram realizadas várias reuniões com diferentes elementos da empresa, nas quais se promoveram debates abertos e dinâmicos sobre como aplicar a técnica de Privacidade Diferencial no seu contexto específico. Estas interações permitiram explorar o tema de forma detalhada, identificar as principais funcionalidades desejadas para a solução final e clarificar as restrições técnicas que poderiam vir a ser consideradas.

Após várias semanas de análise e discussão de diferentes abordagens, chegou-se à proposta final do projeto: o desenvolvimento de uma ferramenta capaz de intercetar pedidos feitos a APIs e aplicar mecanismos de Privacidade Diferencial às respostas que contenham dados sensíveis.

Com a solução finalmente definida, foi então possível definir de forma objetiva alguns dos requisitos funcionais que a mesma deveria apresentar. Entre os mais relevantes, destacam-se:

- Interceção e modificação de respostas de APIs para aplicação de ruído diferencial;
- Suporte a diferentes mecanismos de Privacidade Diferencial;
- Sistema de gestão e controlo do orçamento de privacidade;
- Possibilidade de configuração, permitindo ao utilizador escolher os parâmetros desejados;
- Auditoria das configurações e dos resultados obtidos.

Para além dos requisitos funcionais, foram definidas três características que representam os requisitos não funcionais que a ferramenta deverá cumprir, isto é, os atributos que o sistema deve apresentar para garantir o funcionamento e a integração com o ambiente em que será utilizado:

- **Configurabilidade:** A solução deve ser o mais generalizada e configurável possível para que possa ser utilizada em diferentes contextos sem a necessidade de modificações extensas. A ideia passa por criar um produto flexível,seguindo a filosofia de design "plug and play" [33], que permita uma fácil adaptação às diversas necessidades que vão sendo encontradas.
- **Escalabilidade:** A solução deve ser capaz de crescer e adaptar-se às diferentes necessidades de uso sem perder a sua capacidade de desempenho. Para isso, deverá ser adotada uma arquitetura modular baseada em containers Docker, de modo a garantir maior flexibilidade e facilidade de manutenção e uma rápida adaptação a diferentes ambientes. Além disso, esta base em Docker permite uma futura migração para Kubernetes, facilitando a gestão de múltiplos containers e oferecendo ainda mais elasticidade na utilização destes recursos.
- **Alinhamento Tecnológico:** A solução deve utilizar, sempre que possível, tecnologias e ferramentas já conhecidas pela equipa da empresa. Esta abordagem facilita a sua utilização e permite uma integração mais simples com as práticas já estabelecidas na empresa. Apesar de não ser um critério obrigatório, é um ponto a considerar aquando da escolha das tecnologias.

4.2 Seleção de Ferramentas e Tecnologias

Concluída a fase de levantamento de requisitos e a definição das funcionalidades da solução, segue-se uma etapa essencial em qualquer projeto a ser desenvolvido: a escolha criteriosa das ferramentas e tecnologias a serem utilizadas ao longo de toda a implementação.

As principais ferramentas utilizadas ao longo do projeto são apresentadas e descritas a seguir, organizadas em quatro categorias de acordo com a sua função nesta etapa de desenvolvimento:

(A) Gestão e Base de Dados;

(B) Infraestrutura e Cache;

(C) Framework e Biblioteca;

(D) Linguagem de Programação.

Gestão e Base de Dados	
PostgreSQL	Sistema de gestão de bases de dados relacionais open-source, reconhecido pela sua robustez, segurança e conformidade com os padrões SQL. É escalável, extensível e estável, suportando tanto dados relacionais como dados não relacionais. Neste contexto será utilizado para criar a base de dados do Módulo de Privacidade Diferencial, armazenando todas as diferentes configurações e opções necessárias ao correto funcionamento da técnica.
Flyway	Ferramenta utilizada para a migração e controlo de versões da base de dados. Permite aplicar scripts SQL de forma incremental, segura e automatizada, facilitando o rastreamento das diferentes alterações efetuadas. Neste projeto, esta ferramenta foi utilizada para gerir e executar múltiplas modificações estruturais na base de dados do Módulo de Privacidade Diferencial.

Tabela 4.1: Categoria C - Gestão e Base de Dados

Infraestrutura e Cache	
Docker	Plataforma que permite aos desenvolvedores construir, implementar, executar, atualizar e gerir containers, isto é, unidades que encapsulam uma aplicação e todas as suas dependências. Optou-se por utilizar containers para cada um dos componentes da solução de forma a garantir características essenciais como isolamento, facilidade de manutenção e portabilidade.
Traefik	Proxy reverso projetado para simplificar o encaminhamento de pedidos para diferentes serviços de backend. Atua como ponto de entrada para os pedidos, direcionando-os de forma dinâmica para os serviços adequados, integrando-se com diversas ferramentas, como o Docker. No projeto, foi utilizado para gerir a circulação de dados entre os componentes, principalmente na gestão da comunicação dos mesmos com o middleware.
Redis	Base de dados em memória utilizada para armazenar dados temporários de forma rápida. É usada como cache de segundo nível (L2) para acelerar as aplicações, reduzindo os tempos de resposta e melhorando a performance geral do sistema. Recorremos a esta abordagem para minimizar o tempo de resposta da implementação, mantendo em cache as respostas já com o ruído aplicado, provenientes do Módulo de Privacidade Diferencial.

Tabela 4.2: Categoria D - Infraestrutura e Cache

Framework e Biblioteca	
FastAPI	Framework web utilizado para criar APIs com Python. Projetado para ser simples e intuitivo, permite aos desenvolvedores construir aplicações de forma rápida e eficiente. No nosso projeto, foi utilizado para desenvolver a API principal do sistema, responsável pelos endpoints que fazem a comunicação entre o Módulo de Privacidade Diferencial e o novo middleware.
Diffprivlib	Biblioteca open-source, em Python, desenvolvida pela IBM, dedicada à técnica de Privacidade Diferencial e Machine Learning. Oferece um amplo conjunto de mecanismos (2.8), além de um contador de orçamento de privacidade (2.6) que facilitam a implementação dos algoritmos pretendidos. Foi utilizada para auxiliar na aplicação de ruído nos dados sensíveis intercetados.

Tabela 4.3: Categoria B - Framework e Biblioteca

Linguagem de Programação	
Python	Linguagem de programação de alto nível conhecida pela sua simplicidade e legibilidade, amplamente usada em desenvolvimento web, automação, testagem de software, análise de dados e Machine Learning. Esta foi a linguagem utilizada para a implementação completa do Módulo de Privacidade Diferencial.
Go	Linguagem de programação criada pela empresa Google, sendo conhecida pela sua eficiência, simplicidade e bom desempenho em sistemas concorrentes e distribuídos. Esta linguagem foi utilizada para o desenvolvimento do middleware (plugin) responsável pela interceção de pedidos e das respetivas respostas.

Tabela 4.4: Categoria A - Linguagem de Programação

4.3 Desenvolvimento na Trust Systems

Para se compreender melhor como abordei a solução proposta, é fundamental ter-se um conhecimento prévio de como funciona todo o processo de desenvolvimento em qualquer projeto da Trust Systems. Todos os projetos desenvolvidos pela mesma seguem um método rigoroso e padronizado que engloba diferentes ambientes de trabalho. Assim, o fluxo de procedimento é estruturado e dividido em quatro ambientes distintos, onde cada um apresenta uma função e finalidade específica, conforme demonstrado na figura 4.1, numerados de 1 a 4, de acordo com a ordem do processo:

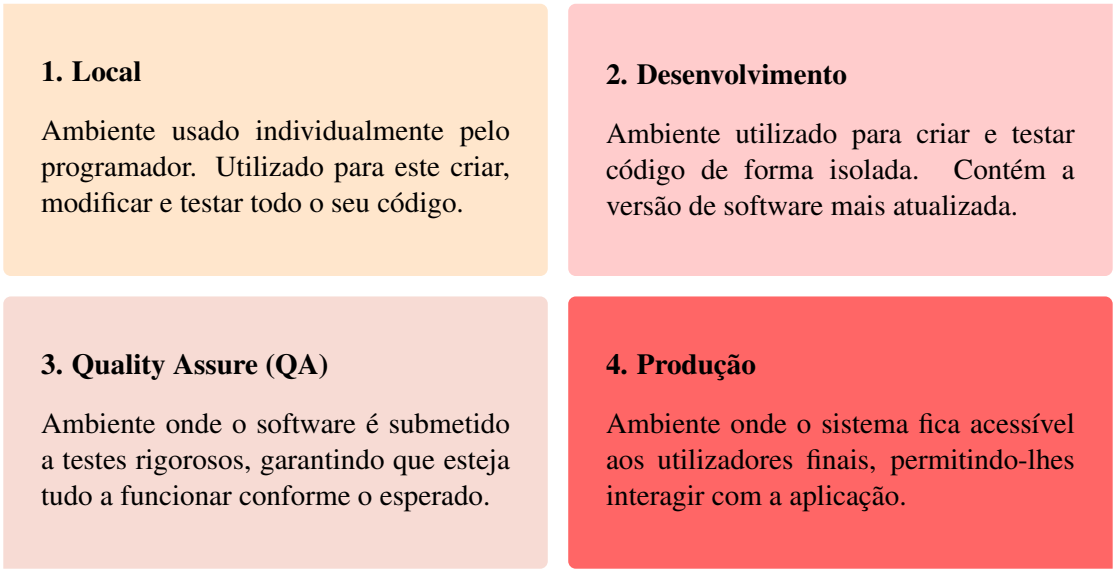

Figura 4.1: Ambientes de desenvolvimento

Todos os ambientes descritos na figura 4.1 são compostos por conjuntos de containers Docker, isto é, pequenos "pacotes" virtuais que permitem criar e executar aplicações de forma isolada e que encapsulam cada uma das partes dessa aplicação, facilitando a sua escalabilidade e portabilidade.

A comunicação entre estes containers é gerida por um outro container que executa um componente fundamental: o Traefik, uma ferramenta que atua como Proxy Reverso, funcionando como ponto de entrada de todos os pedidos externos e encaminhando-os, de forma eficiente e automatizada, para os serviços internos adequados, conforme as regras e labels dinâmicas definidas no Docker Compose, isto é, no ficheiro que permite configurar todos os containers de forma simples.

A Figura 4.2 ilustra um exemplo de um ambiente de uma aplicação da Trust Systems onde, numa máquina de produção, está a correr um container que contém o Traefik. Este atua como canal de entrada, escutando, geralmente, nas portas padronizadas 80 (HTTP) e 443 (HTTPS). Dentro da rede gerida pelo Traefik encontram-se os serviços que compõem a aplicação (como o Frontend, Backend e as respetivas Bases de Dados), todos executados em containers distintos.

O Traefik é, portanto, uma peça-chave na infraestrutura da Trust Systems, funcionando como o único ponto de entrada para todos os pedidos externos. Toda a comunicação é centralizada nele, o que garante maior controlo e segurança. Funcionalidades como deteção automática de serviços, configuração dinâmica, gestão de certificados HTTPS e suporte nativo a containers Docker são alguns dos principais fatores que tornam o Traefik a escolha ideal da empresa para esta arquitetura.

Compreendido o papel fulcral que o Traefik apresenta, é importante entender na prática como funciona todo este fluxo interno de pedidos. Esta compreensão é fundamental para que, posteriormente, se possa realizar uma integração correta e eficaz de novos componentes nesta arquitetura para a implementação da nossa solução, que incluirá a integração de um novo componente res-

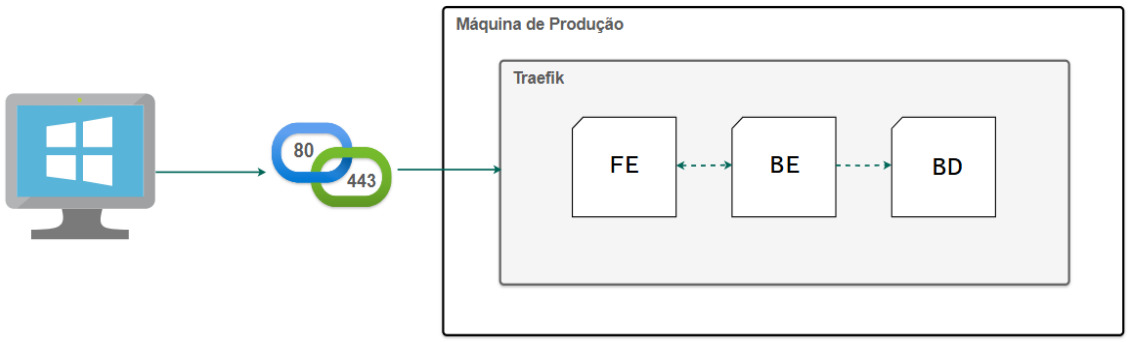

Figura 4.2: Ambiente de uma Aplicação Trust Systems

ponsável pela interceção e manipulação de pedidos e respetivas respostas.

De forma simplificada, podemos visualizar na figura 4.3 esse mesmo fluxo, num cenário sem qualquer componente adicional. Todo este processo inicia-se com o utilizador a realizar uma chamada HTTP para um URL específico da aplicação em questão (1). Esse pedido é então recebido pelo Traefik que, através de labels dinâmicas associadas aos serviços consegue encaminhar o pedido para o container correto (2). A partir desse momento, esse serviço processa internamente a lógica que é necessária dando origem a uma resposta, sendo essa seguidamente devolvida ao Traefik (3) que a reencaminha finalmente para o utilizador inicial (4).

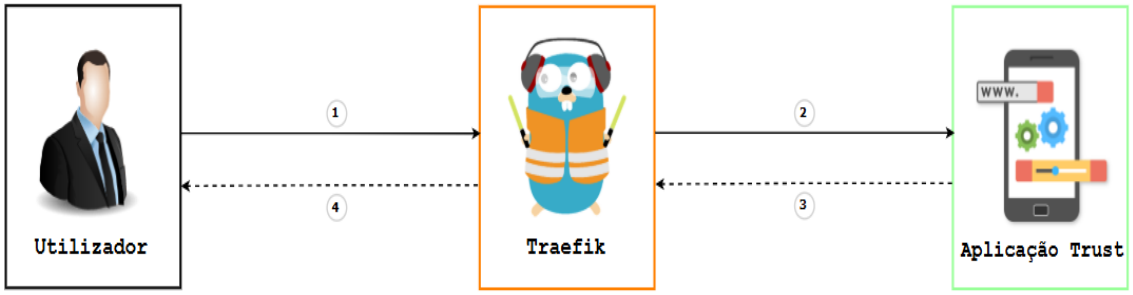

Figura 4.3: Fluxo de Pedido com Traefik

É importante salientar que o Traefik apenas funciona como elo de ligação entre o utilizador e os serviços, sem interferir ou modificar o conteúdo dos pedidos ou das respostas. No entanto, embora este seja o comportamento padrão, o mesmo pode ser configurado para realizar alterações, como modificar cabeçalhos HTTP ou, por exemplo, reescrever caminhos. Essas transformações são feitas por meio da introdução de novos elementos designados de middlewares, isto é, componentes capazes de intercetar e adaptar pedidos e respostas antes que cheguem ao destino final.

O conhecimento prévio destes conceitos é indispensável, uma vez que é precisamente no percurso representado na figura 4.3, e através da introdução de um middleware nesse fluxo, que será

possível cumprir um dos primeiros requisitos do projeto: a interceção de pedidos e respostas.

4.4 Visão Geral e Arquitetura

Agora que se encontram definidas as bases essenciais para o nosso projeto, nomeadamente o levantamento de requisitos, a seleção criteriosa de ferramentas e tecnologias a utilizar, bem como a análise prévia de como funciona o fluxo de desenvolvimento de um projeto na Trust Systems, estamos prontos para avançar para a implementação prática da solução proposta.

Tal como descrito inicialmente, o objetivo consiste na elaboração de uma ferramenta capaz de intercetar pedidos e aplicar a técnica de Privacidade Diferencial nas suas respostas. Esta solução funcionará como uma camada intermédia entre o utilizador e todos os serviços de uma determinada aplicação, com intuito de acrescentar um nível extra de privacidade nos dados sensíveis.

Como tal, é possível decompor a nossa solução em dois elementos principais que, numa fase inicial, poderão ser abordados de forma independente. Esses dois componentes são:

Middleware de Interceção de Pedidos	**Módulo de Privacidade Diferencial**
Componente localizado entre o utilizador e os serviços da aplicação que tem como objetivo capturar, modificar e redirecionar pedidos e as respetivas respostas.	Componente autónomo dedicado à correta aplicação dos mecanismos de proteção diferencial, devolvendo respostas com ruído aos utilizadores.

Embora ambos possam ser tratados e desenvolvidos de forma individual, permitindo uma evolução independente e isolada de cada parte, estes devem, obviamente, funcionar de forma articulada e coesa. É, por isso, fundamental compreender de que forma estes componentes se interligam, uma vez que apenas com o sucesso de ambos será possível obter uma solução eficaz.

Posto isto, é essencial compreender a dinâmica desta nova arquitetura, agora renovada com dois novos elementos. A figura 4.4 apresenta uma representação esquemática da solução proposta, ilustrando detalhadamente todos os novos itens, os percursos de comunicação entre eles, bem como o fluxo completo de qualquer pedido. Esta nova abordagem apresenta diversos pontos desafiantes tecnicamente que foram enfrentados para garantir o sucesso desta implementação.

Tal como é possível verificar, tanto o percurso do pedido bem como o modo como a resposta é devolvida ao utilizador sofrem agora algumas alterações relativamente ao fluxo existente (Figura 4.3). Nesta nova solução, o utilizador faz um pedido à aplicação Trust. Este pedido é inicialmente recebido pelo Traefik que continua a atuar como Proxy Reverso (1). No entanto, antes de ser encaminhado diretamente para a aplicação, o pedido passa agora por um novo middleware personalizado, introduzido nesta arquitetura agora reformulada (2). Esse middleware desempenha um papel crucial, permitindo análises preliminares e possíveis alterações na estrutura da resposta, com o objetivo de a preparar adequadamente para a sua receção ao módulo de Privacidade Diferencial. Passando essa fase intermédia, o pedido chega à aplicação Trust (3), que o processa e gera

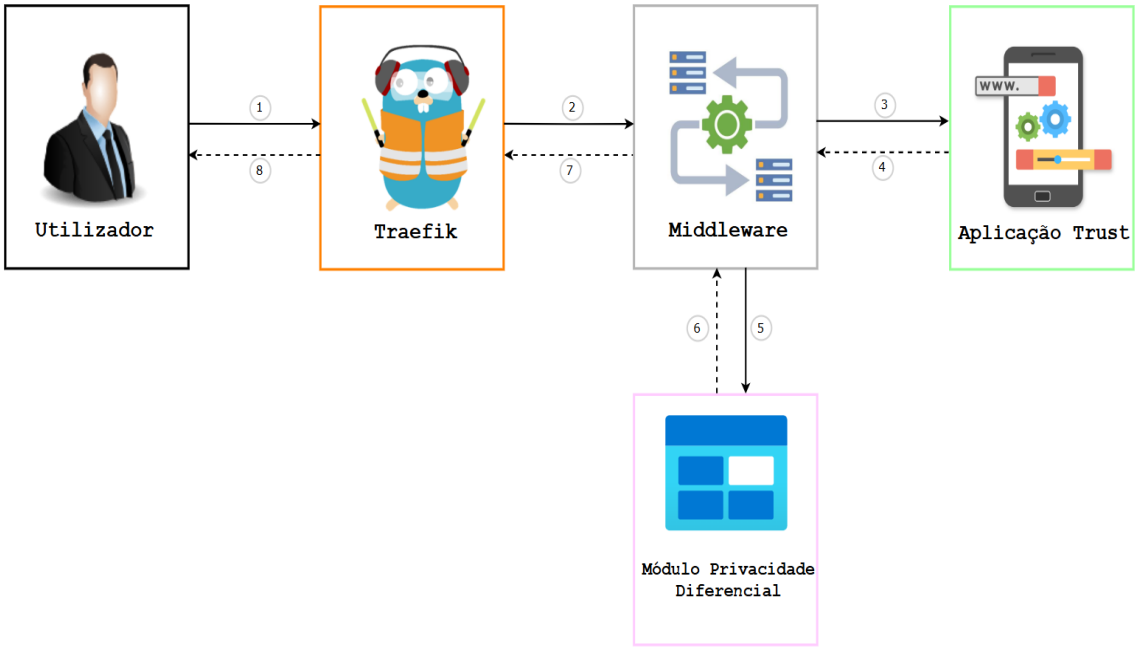

Figura 4.4: Arquitetura Inicial Proposta

a resposta correspondente. Ao contrário do que até então acontecia, essa resposta não é devolvida de imediato ao utilizador, uma vez que é agora intercetada pelo middleware (4) e reencaminhada para o novo módulo de Privacidade Diferencial através da sua API (5). Esse módulo aplica os procedimentos necessários de acordo com as configurações específicas da implementação do próprio módulo. Após toda a aplicação da técnica, a resposta, agora já com a devida proteção, regressa ao middleware (6), passando depois pelo Traefik (7) que, finalmente, a devolve ao utilizador (8).

Este fluxo permite executar de forma eficaz todos os objetivos inicialmente estabelecidos, possibilitando não apenas a interceção de pedidos e respostas, mas também a aplicação de ruído nas mesmas através do Módulo de Privacidade Diferencial. Contudo, durante uma análise mais pormenorizada do fluxo representado e após algumas reuniões junto dos membros da equipa da Trust Systems, levantaram-se algumas questões relativamente ao impacto que a introdução de um novo middleware poderia trazer ao sistema. Sendo um processo que ocorre em tempo real, a presença deste novo componente poderia vir a provocar um aumento na latência das respostas, comprometendo a velocidade da mesma e, consequentemente, a experiência do utilizador.

Como tal, para mitigar esse problema, recorreu-se a um instrumento já utilizado em muitos projetos da empresa, o Redis. A introdução desta camada de cache possibilita à aplicação guardar e recuperar dados com uma latência muito reduzida, tirando assim proveito das vantagens deste armazenamento em memória. O Redis permite também configurar um tempo de expiração para os dados que estão em cache, garantindo que a informação nela presente seja renovada periodicamente. No contexto da nossa implementação, foi definido um tempo de expiração de cinco minutos, valor escolhido por representar um equilíbrio adequado entre a atualização dos dados e a redução da carga sobre a aplicação em causa.

A lógica desta nova adaptação é simples: após o middleware receber o pedido e antes de o encaminhar para a aplicação Trust, é feita uma verificação na cache Redis para averiguar se já existe uma resposta previamente processada e armazenada para aquele pedido específico. Se a resposta já existir, é devolvida ao utilizador, estando previamente protegida pelo módulo. (Opção A). Por outro lado, se não existir nenhuma entrada correspondente, o pedido segue o fluxo inicialmente estabelecido (Opção B). A diferença, neste caso, reside no facto de que, após a aplicação de ruído na resposta, esta é armazenada no Redis, ficando disponível para futuros pedidos idênticos.

Esta nova e última alteração à arquitetura pode ser visualizada na figura 4.5, onde agora sim está representada a abordagem final implementada. Esta modificação permitiu alinhar os objetivos propostos com um não decréscimo no desempenho do sistema (Resultados no Capítulo 5).

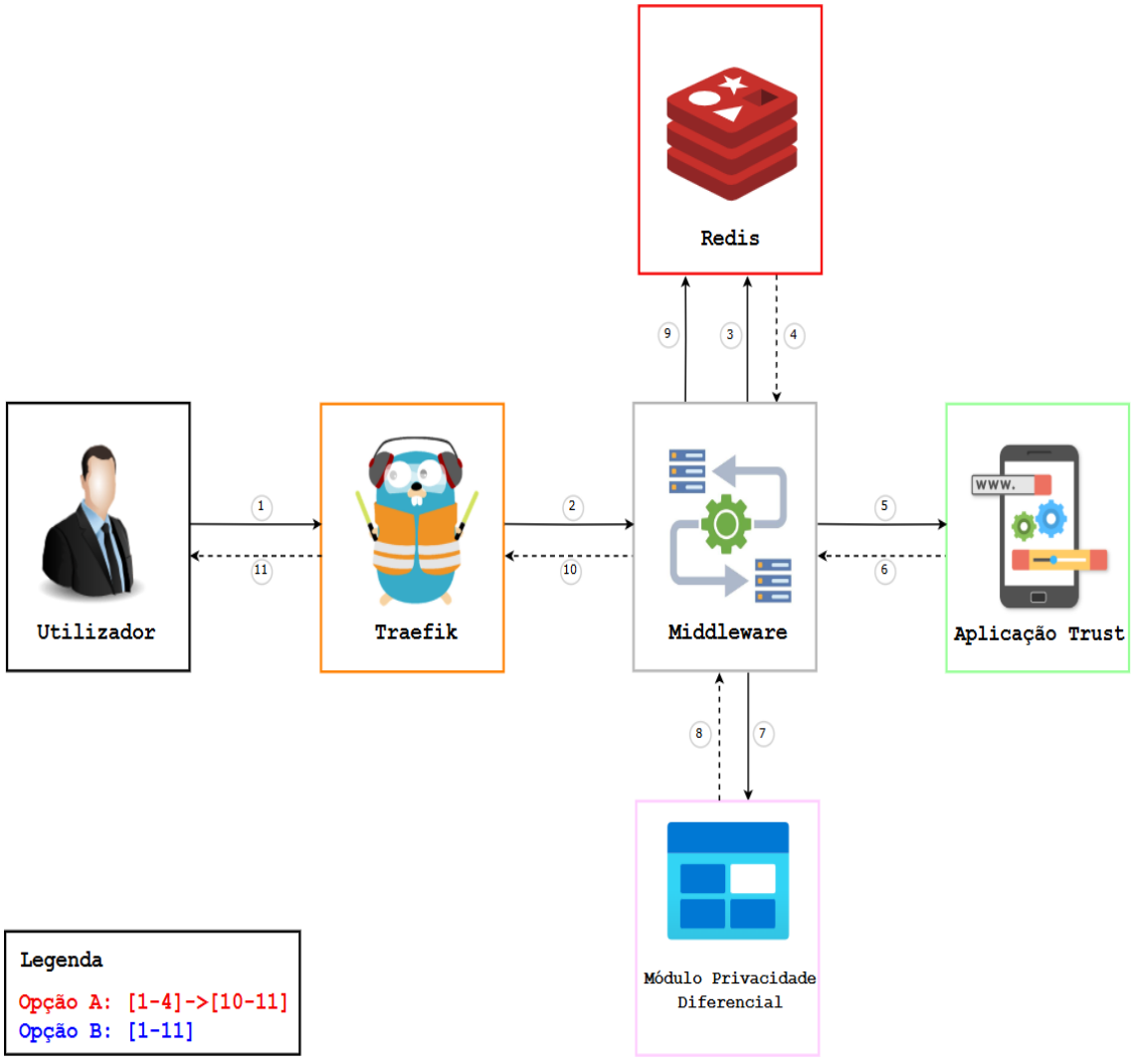

Figura 4.5: Arquitetura Final Implementada

Tendo agora definida, estruturalmente, a solução final, podemos avançar para uma descrição mais individual de cada elemento. Nas secções seguintes, cada um destes novos componentes

(Middleware e Módulo de Privacidade Diferencial) será analisado em maior detalhe, sendo abordadas as decisões técnicas tomadas e todos os desafios ultrapassados até chegar ao objetivo final.

4.5 Middleware de Interceção de Pedidos

No início desta implementação, tornou-se evidente a necessidade de intercetar pedidos e respostas trocadas entre utilizadores e uma determinada aplicação, com o objetivo de conseguir aplicar Privacidade Diferencial nos dados devolvidos pela mesma. Como tal, para se concretizar esse requisito, ficou claro que seria necessário inserir uma camada intermédia entre o utilizador e a aplicação em questão, capaz de manipular diretamente o conteúdo das respostas HTTP.

4.5.1 Limitações das Soluções Existentes

Tendo sido adotado o Traefik como Proxy Reverso na arquitetura da solução, a primeira abordagem consistiu em explorar as funcionalidades oferecidas por esta ferramenta, nomeadamente os seus middlewares padrão. Contudo, rapidamente se concluiu que os middlewares existentes apenas atuavam ao nível dos pedidos HTTP propriamente ditos, não oferecendo a capacidade de modificar o corpo da resposta, funcionalidade indispensável para a concretização deste projeto.

Perante esta limitação, avançou-se para uma segunda abordagem: a exploração e análise do Catálogo de Plugins disponibilizado pelo Traefik. Esse catálogo é composto por inúmeros plugins desenvolvidos pela comunidade de utilizadores que contribuem para esta ferramenta. De entre essas contribuições, destaca-se a criação de plugins personalizados, que são partilhados publicamente e, posteriormente, mantidos pela equipa do Traefik.

Esses plugins funcionam como um componente adicional que se integra com o Traefik para estender ou modificar as suas funcionalidades base, permitindo lidar com cenários mais específicos e personalizados. Desta forma, o catálogo funciona como um repositório onde qualquer utilizador ou desenvolvedor pode disponibilizar um plugin. Esses plugins permitem resolver casos específicos, que podem ser úteis para muitos outros utilizadores com necessidades semelhantes.

Nesta fase, a expectativa era encontrar um plugin já existente que pudesse ser reutilizado ou adaptado para o nosso caso específico. Contudo, após uma análise cuidada, verificou-se que, mais uma vez, nenhum dos plugins disponíveis oferecia o requisito essencial de manipulação do corpo da resposta HTTP. Os plugins mais relevantes identificados operavam apenas ao nível do próprio pedido, modificando os seus cabeçalhos ou interferindo no controlo de rotas dos pedidos. Na Tabela 4.5 são apresentados alguns desses plugins identificados durante esta análise, acompanhados de uma breve descrição e limitação que levou à sua exclusão das opções selecionáveis.

Face a todas estas limitações enfrentadas, chegámos a um impasse sobre como avançar. Após várias reuniões com elementos da equipa da Trust Systems, concluímos que a única solução viável seria desenvolver e implementar um plugin personalizado, escrito em Go, linguagem padrão suportada pelo Traefik para este tipo de extensões. Embora a construção de um plugin do zero

representasse um esforço técnico muito complexo, essa abordagem revelou-se a única forma de conseguir atingir os objetivos do projeto.

NOME DO PLUGIN	DESCRIÇÃO
Rewrite Body[1]	Reescreve o corpo de uma resposta HTTP, substituindo uma expressão regular por uma string de substituição (por exemplo, trocar todas as ocorrências de "foo" por "bar"). Sendo limitado a expressões regulares, é uma solução muito restrita.
Rewrite Body (with compression support)[2]	Semelhante ao Rewrite Body, realiza substituições com expressões regulares, mas com mais funcionalidades e suporte a novas e diferentes condições (por exemplo, permite alterações apenas se o código HTTP for igual a 200). Embora mais flexível, ainda não é suficiente para o objetivo proposto.
Rewrite Headers[3]	Substitui cabeçalhos de respostas HTTP utilizando expressões regulares (por exemplo, no cabeçalho, trocar todos os URLs que comecem por http:// por https://). Aplicado apenas ao nível do cabeçalho, foi uma alternativa logo descartada.

Tabela 4.5: Comparação entre Plugins do Traefik

4.5.2 Estrutura e Configuração do Plugin Personalizado

Tendo chegado à conclusão de que a única opção seria a construção de um novo plugin, o passo seguinte concentrou-se em compreender como essa implementação poderia ser realizada e de que forma se podia executá-la com base na arquitetura da solução e na infraestrutura da empresa.

Conforme descrito na secção 4.3, a Trust Systems utiliza o Traefik como Proxy Reverso. Deste modo, a primeira fase centrou-se na análise dos recursos oferecidos por essa ferramenta para a criação de novos plugins. Verificou-se que o Traefik disponibilizava suporte a plugins locais, isto é, pequenas unidades em código Go que permitem alterar o comportamento padrão da ferramenta sem a necessidade de alterar diretamente o código-fonte original.

A utilização de plugins locais torna possível a adição de novas funcionalidades de forma rápida e isolada, facilitando a realização de testes e garantindo a segurança e privacidade do código dentro da empresa. Aproveitando essa funcionalidade, desenvolvemos um plugin denominado `applydp`. A sua integração foi feita no ambiente local onde o Traefik é configurado, mais especificamente em – `traefik-infraxs-local` – onde criámos a pasta `plugins-local`, seguindo as recomendações da documentação da ferramenta para a criação de plugins locais.

[1] https://plugins.traefik.io/plugins/62947307108ecc83915d7783/rewrite-body
[2] https://plugins.traefik.io/plugins/6294728effc0cd18356a97c3/rewrite-body-with-compression-support
[3] https://plugins.traefik.io/plugins/628c9eb3ffc0cd18356a979b/rewrite-http-response-headers-for-traefik

Após uma consulta e análise minuciosa da documentação oficial, procedeu-se à organização da estrutura de pastas e ficheiros de forma a cumprir integralmente com as práticas recomendadas. A estrutura final do plugin ficou definida conforme ilustrado na Listagem 4.1:

```
C:\devenv\traefik\traefik-infraxs-local\plugins-local\

|-- src
    |-- github.com
        |-- traefik
            |-- applydp
                |-- apply.go
                |-- go.mod
                |-- go.sum
                |-- .traefik.yml
```

Listagem 4.1: Estrutura de Pastas e Ficheiros do Novo Plugin

Como é possível verificar, a nova pasta `plugins-local` contém outras pastas organizadas hierarquicamente, começando por `src -> github.com -> traefik`, e, finalmente, uma com o nome do plugin: `applydp`. Dentro desta última encontram-se quatro ficheiros distintos:

- `apply.go`: Ficheiro responsável por toda a lógica do plugin;
- `go.mod`: Ficheiro responsável por controlar as suas dependências e versões;
- `go.sum`: Ficheiro que contém os checksums das dependências do `go.mod`;
- `.traefik.yml`: Ficheiro que funciona como documentação base do plugin.

De salientar que tanto o ficheiro `go.mod` como o ficheiro `go.sum` são gerados automaticamente após a compilação do `apply.go` e, por esse motivo, estes ficheiros podem sofrer alterações constantes à medida que o código deste último evolui.

Para que toda esta estrutura de ficheiros funcione corretamente, é necessário ajustar determinadas configurações de alguns ficheiros padrão do Traefik, nomeadamente o `dynamic.yml`, o `traefik.yml` e o `docker-compose.yml`. Este último deve-se ao facto de o Traefik correr dentro de um ambiente Docker, exigindo configurações específicas para o seu correto funcionamento. Segue-se a descrição de cada ficheiro, bem como os ajustes efetuados em cada um deles:

`Ficheiro traefik.yml`: Responsável por toda a configuração estática do Traefik, isto é, tudo o que precisa de estar definido até ao momento em que o serviço é iniciado. Especifica opções como as portas de entrada (*entrypoints*), os provedores (*providers*, de onde virá a configuração dinâmica), o painel de controlo (*dashboard)* e, também, os plugins locais. Como é possível verificar em 4.2, foi incluída a definição do novo plugin na secção `experimental.localPlugins`:

```
1 entryPoints:
2   web:
3     address: ":80"
4   websecure:
5     address: ":443"
6
```

```
api:
  dashboard: true

providers:
  file:
    filename: /etc/traefik/dynamic.yml

experimental:
  localPlugins:
    applydp:
      moduleName: github.com/traefik/applydp
```

Listagem 4.2: Configuração Estática do Traefik - Ficheiro traefik.yml

`Ficheiro dynamic.yml`: Responsável por toda a configuração dinâmica do Traefik, isto é, todas as regras de funcionamento que podem ser modificadas sem reiniciar o Traefik. Essas regras incluem a definição de rotas, serviços e, também, middlewares. Como podemos ver em 4.3, foi adicionado o middleware `applydp`, usando o plugin definido no ficheiro estático `traefik.yml`:

```
http:
  routers:
    api:
      entryPoints:
        - websecure
      service: api@internal

  middlewares:
    applydp:
      plugin:
        applydp: {}
```

Listagem 4.3: Configuração Dinâmica do Traefik - Ficheiro dynamic.yml

`Ficheiro docker-compose.yml`: Responsável por definir e configurar os serviços a serem executados, sendo alterado de modo a garantir que o Traefik reconhece corretamente o novo plugin local. Para tal, é necessário indicar o nome do módulo do plugin através dos comandos associados ao serviço do Traefik. Em 4.4, observamos de que forma esse comando é adicionado:

```
traefik:
  command:
  --experimental.localPlugins.applydp.moduleName=github.com/traefik/applydp
```

Listagem 4.4: Configuração do Traefik - Ficheiro docker-compose.yml

Estando o plugin corretamente definido e disponível, é agora necessário aplicá-lo à aplicação cujos pedidos HTTP queremos intercetados e processados pelo plugin. Esta última etapa é feita através da definição de labels no `docker-compose.yml` da aplicação, associando o middleware ao *router* correspondente. No exemplo 4.5, é mostrado como deve ser configurada essa *label*, substituindo <*router-name*> pelo nome real do *router* no contexto da aplicação:

```
labels:
  - traefik.http.routers.<router-name>.middlewares.applydp@file
```

Listagem 4.5: Configuração de Aplicação Exemplo

4.5.3 Implementação do Plugin Applydp

Após preparação e definição da estrutura de todos os ficheiros necessários para o correto funcionamento do novo plugin, avançámos depois para a implementação do ficheiro `apply.go`, já mencionado na secção anterior. Este ficheiro contém o código completo em linguagem Go (linguagem padrão para o desenvolvimento de plugins no Traefik), e é responsável por gerir o fluxo do processo de interceção de pedidos e respostas, desde o momento em que o pedido é recebido até que a resposta seja entregue ao utilizador. Em 4.6, 4.7, 4.8, 4.9 e 4.10, apresentam-se os excertos de código mais relevantes que compõem o ficheiro `apply.go`. Cada um destes excertos corresponde a uma etapa fundamental no desenvolvimento do plugin, que será analisada detalhadamente com base no respetivo código. De referir que, para além destes trechos de código, existem outras partes omitidas, que podem ser consultadas nos Anexos, onde se encontra o ficheiro completo.

Etapa 1: Estrutura Inicial e Configuração do Plugin

Esta etapa consiste na criação da estrutura base do plugin, permitindo a sua correta integração com o Traefik. Nesta fase é definida a estrutura *ApplyDP*, que armazena os elementos necessários ao funcionamento do plugin: o nome, as configurações atribuídas, o endereço do microserviço de cache e o chamado *Next Handler*. Este último representa o próximo passo no processamento do pedido, ou seja, depois do plugin executar toda a sua lógica, o pedido continua o seu percurso normal, sendo encaminhado para o componente seguinte. Por fim, a função *New* é responsável por inicializar o plugin, sendo aí obtido o respetivo endereço do microserviço de cache.

```
type ApplyDP struct {
    next            http.Handler
    name            string
    config          *Config
    cacheServiceURL string }

func New(ctx context.Context, next http.Handler, config *Config, name string) {
    cacheURL := os.Getenv("CACHE_SERVICE_URL")
    if cacheURL == "" {
        cacheURL = "http://host.docker.internal:8011"
    }
    return &ApplyDP{next, name, config, cacheURL}, nil }
```

Listagem 4.6: Etapa 1 - Estrutura Inicial e Configuração do Plugin

Etapa 2: Verificação do Redis

Quando o utilizador realiza um pedido, o plugin inicia a sua atividade. A sua primeira tarefa consiste em averiguar se já existe uma resposta previamente armazenada em cache para aquele pedido específico. O objetivo inicial passava por fazer o plugin comunicar diretamente com o Redis, contudo, devido a restrições de segurança impostas pela criação de plugins locais do Traefik, tal abordagem mostrou-se inviável. Como tal, a solução passou pela criação de um microserviço dedicado exclusivamente à gestão da cache. Assim, o plugin envia o pedido a esse microserviço, contendo informação importante como o método HTTP, o caminho e dados associados. Caso exista uma resposta correspondente, esta é devolvida ao utilizador, evitando a execução de passos desnecessários. Pelo contrário, se não houver nenhuma correspondência, todo o fluxo prossegue.

```
cacheGetPayload := CacheGetRequest{req.Method, req.URL.Path, requestData}
resp, _ := http.Post(a.cacheServiceURL+"/cache/get", "application/json", bytes.
    NewReader(payload))

if resp.StatusCode == http.StatusOK {
    json.NewDecoder(resp.Body).Decode(&cacheResult)
    if cacheResult.Cached {
        rw.Write(cachedData)
        return
    }
}
```

Listagem 4.7: Etapa 2 - Verificação do Redis

Etapa 3: Encaminhamento do Pedido à Aplicação e Processamento da Resposta

Se, na etapa 2, a resposta não for encontrada na cache, avança-se para este novo passo. Nesta fase, tendo o pedido sido previamente preparado, este é enviado para a aplicação de destino. Após receber a resposta, verifica-se se a mesma se encontra comprimida, isto é, se o seu tamanho foi reduzido de modo a acelerar a chegada da resposta ao destino. Se for esse o caso, o plugin procede com a sua descompressão, permitindo que todo o seu contéudo possa ser devidamente analisado.

```
targetURL := fmt.Sprintf("https://%s%s", req.Host, req.URL.RequestURI())
respOriginal, _ := http.DefaultClient.Do(req)
body, _ := decompressIfGzip(respOriginal)
```

Listagem 4.8: Etapa 3 - Encaminhamento do Pedido à Aplicação e Processamento da Resposta

Etapa 4: Aplicação de Privacidade Diferencial

Com a resposta já descomprimida, é possível analisar todo o conteúdo da mesma. A próxima etapa passa por preparar a devida resposta para ser enviada para o Módulo de Privacidade Diferencial, organizando e estruturando todos os dados num formato JSON específico. A esses dados recebidos, o plugin vai adicionar dois novos elementos importantes que não fazem parte da resposta originalmente recebida: o caminho do pedido (*endpoint*) e o método HTTP utilizado (GET, POST,

PUT, ou DELETE). Todos estes elementos são necessários uma vez que representam detalhes importantes para decisões que serão posteriormente tomadas no Módulo de Privacidade Diferencial.

Para tornar o processo mais claro, consideremos um exemplo: um pedido GET para o *endpoint /users/1*, que retorna informações pessoais do utilizador 1. Nesta fase, e antes de ser enviada ao componente seguinte, a resposta é transformada conforme ilustrado abaixo:

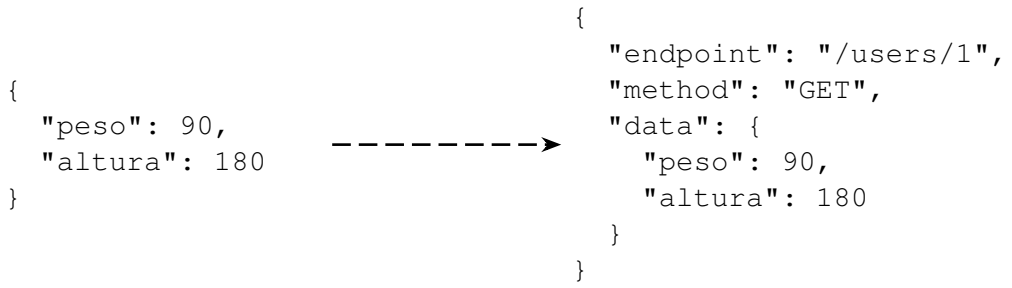

Figura 4.6: Exemplo de Pedido GET após Etapa 4

Após a construção deste novo JSON, os dados são enviados ao módulo, onde são processados de acordo com as configurações e regras definidas para garantir a devida proteção dos mesmos.

```
dpPayload := map[string]interface{}{
        "endpoint": req.URL.Path,
        "method":   req.Method,
        "data":     data,
    }

respDP, _ := http.Post("http://api:8000/apply_dp", "application/json", bytes.
    NewReader(dpJson))
dpBody, _ := io.ReadAll(respDP.Body)
```

Listagem 4.9: Etapa 4 - Aplicação de Privacidade Diferencial

Etapa 5: Armazenamento no Redis e Resposta ao Utilizador

Aplicados os mecanismos de Privacidade Diferencial, obtém-se a resposta final, agora devidamente protegida. Esta resposta, quando obtida, é devolvida ao plugin que, num primeiro momento, a armazena no microserviço de cache. Com esse passo concretizado, garantimos que, quando executados pedidos futuros que sejam idênticos, estes possam ser respondidos ao utilizador de forma imediata, evitando grande parte das etapas descritas. Finalmente, com a resposta protegida e devidamente guardada no Redis, esta é enviada ao utilizador que realizou o pedido. Importa referir que este passo de armazenamento poderá, em trabalhos futuros, ser explorado de forma assíncrona, permitindo que a resposta seja enviada ao utilizador sem depender da conclusão desta operação.

```
cacheSetPayload := CacheSetRequest{
        Method:   req.Method,
        Endpoint: req.URL.Path,
        Data:     requestData,
        Response: json.RawMessage(dpBody),
```

```
        Cached:   true,
    }

http.Post(a.cacheServiceURL+"/cache/set", "application/json", bytes.NewReader(
    cacheSetBody))

rw.Header().Set("Content-Type", "application/json")
rw.Write(dpBody)
```

Listagem 4.10: Etapa 5 - Armazenamento no Redis e Resposta ao Utilizador

4.6 Módulo de Privacidade Diferencial

Ultrapassado o obstáculo da interceção de pedidos e respostas, podemos agora avançar para a aplicação propriamente dita da técnica de Privacidade Diferencial. Conforme descrito no capítulo acerca da Arquitetura da Solução em 4.4, optámos pela implementação de um microserviço independente para esta etapa do projeto, procurando focar esta parte crucial da implementação num módulo específico para esse fim. Todo o processo de desenvolvimento, bem como o funcionamento de todo este módulo, será descrito detalhadamente ao longo desta secção.

4.6.1 Descrição Inicial

Este módulo foi concebido com o propósito de oferecer aos utilizadores uma experiência altamente adaptável e configurável, permitindo-lhes total liberdade para introduzir e ajustar os parâmetros associados à técnica de Privacidade Diferencial, de acordo com as suas necessidades.

A ferramenta é dedicada a qualquer elemento das diversas equipas empresariais com conhecimento contextual da informação a ser analisada, com o intuito de fornecer estatísticas agregadas e informações anonimizadas que contribuam para análises e decisões estratégicas importantes a serem tomadas. Importa reforçar que o objetivo passa por extrair conclusões relevantes sem, em momento algum, comprometer a privacidade individual de qualquer pessoa presente nos dados.

A arquitetura do módulo foi desenvolvida com base em containers Docker, onde cada serviço se encontra isolado num container individual, garantindo características relevantes como portabilidade e escalabilidade. Os serviços são mencionados abaixo e podem ser visualizados em 4.7:

1. API principal (porta 8000) desenvolvida em Python 3.13 e framework FastAPI, responsável por receber a resposta dos pedidos e aplicar todo o processo de anonimização nas mesmas;

2. Base de Dados PostgreSQL (porta 5432), associada à API e gerida através do Flyway, que é encarregue de possíveis migrações que possam vir a ocorrer em todas as diferentes tabelas;

3. Microserviço Redis (porta 8011), conectado com o middleware implementado anteriormente, que tem como objetivo auxiliar no contacto com o servidor Redis usado na empresa.

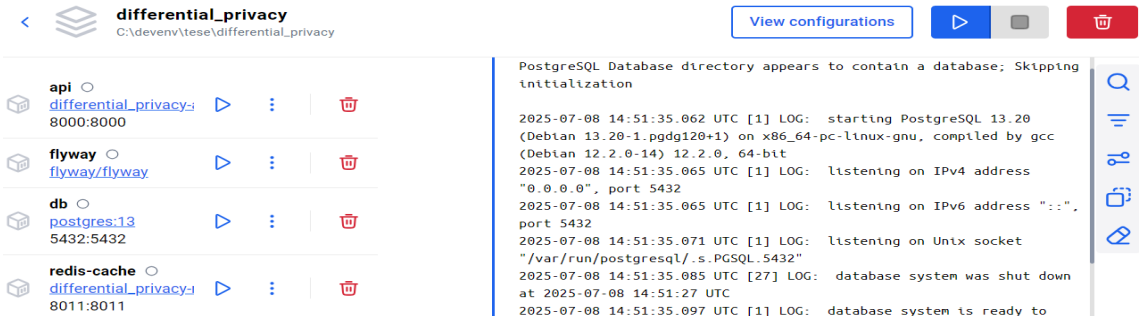

Figura 4.7: Containers Docker - Módulo de Privacidade Diferencial

4.6.2 Modelo de Privacidade Diferencial

Conforme apresentado inicialmente em 2.4, a técnica de Privacidade Diferencial pode ser aplicada segundo duas abordagens distintas: Global ou Local. No nosso caso, a ferramenta implementada foi desenvolvida com base no modelo de Privacidade Diferencial Global, o que significa que esta recolhe e processa os dados originais, sem qualquer deturpação prévia, aplicando posteriormente o ruído no resultado final da consulta/pedido realizado pelo utilizador. Essa perturbação ocorre antes que os dados sejam expostos ao utilizador final, garantindo que os valores exatos e verdadeiros não sejam expostos diretamente.

A principal diferença em relação à definição formal do modelo global reside no facto da nossa ferramenta não ter acesso direto à base de dados. Como já vimos, este módulo processa apenas a resposta a pedidos feitos numa aplicação, tendo sido esse um dos objetivos delineados na fase inicial do projeto. Ao receber essa resposta, a ferramenta vai atuar sobre esses dados nela contidos, aplicando o ruído de acordo com os parâmetros de configuração que o utilizador irá definir futuramente. De resto, o funcionamento mantém-se alinhado com a técnica global formalmente descrita, posicionando a ferramenta como um intermediário entre o utilizador e a aplicação.

Adicionalmente, embora o método seja destinado à proteção de valores estatísticos agregados como, por exemplo, médias ou somas, foi também incorporado um componente extra de anonimização individual aos dados. Assim, além da proteção de resultados agregados, a ferramenta possibilita a anonimização de dados individuais, sejam eles dados numéricos ou categóricos, com base numa nova perspetiva implementada para a aplicação de uma abordagem Local, isto é, o acrescento de uma nova proteção a dados individuais antes de serem expostos ao utilizador final.

Trata-se, contudo, de uma extensão à ferramenta e não de uma implementação tradicional deste modelo, já que não segue o seu procedimento formal. Os valores na base de dados permanecem inalterados, já que não temos acesso à base de dados, e o ruído é aplicado na origem dos dados, logo, no dispositivo do utilizador, e não diretamente na base de dados. Estas características do nosso caso em concreto divergem, assim, do funcionamento previsto do modelo local original.

De forma geral, a solução adota uma abordagem baseada na técnica de Privacidade Diferencial Global, mas incorporando elementos de proteção diferencial local nos dados processados. Essa combinação acrescenta uma camada adicional de proteção, reforçando significativamente a

privacidade na exposição de dados individuais.

4.6.3 Biblioteca Diffprivlib

No capítulo 3, foi efetuado um levantamento e análise das principais ferramentas e frameworks de Privacidade Diferencial disponíveis e que poderiam ser úteis para o nosso projeto. Após uma comparação dos pontos negativos, positivos e necessários para o nosso contexto, chegámos à conclusão de que a ferramenta que mais servia as nossas necessidades seria a biblioteca Diffprivlib.

Desenvolvida pela IBM, International Business Machines, esta biblioteca de Privacidade Diferencial [27] [26] é implementada em Python (compatível com versões superiores à 3.4) e inclui uma série de mecanismos e ferramentas que nos permitem aplicar, de forma simples, este método de anonimização. A Diffprivlib pode ser instalada de forma rápida e direta através do comando *"pip install diffprivlib"*, e faz uso de recursos muito comuns na linguagem Python como, por exemplo, a biblioteca NumPy, muito utilizada para o processamento de dados numéricos.

Como tal, a primeira etapa passou por analisar as funcionalidades disponíveis, selecionando aquelas que nos ajudariam a implementar da melhor forma a nossa solução. Esta biblioteca é composta por dois módulos principais, mecanismos e ferramentas, onde, no primeiro, se encontram as funções que aplicam os diversos Mecanismos Diferenciais e, no segundo, onde se encontram as ferramentas necessárias para simples análises de dados com Privacidade Diferencial.

De forma a mantermos a coerência com o estudo inicial realizado no Capítulo 2, foram selecionados os seguintes mecanismos: Mecanismo de Laplace (A), Mecanismo Gaussiano (B) e Mecanismo Exponencial (C). É importante salientar que a biblioteca utilizada disponibiliza diferentes variantes para cada um desses mecanismos, tendo sido escolhida, em cada caso, a opção que melhor se adequa aos objetivos propostos e mais alinhados com o nosso planeamento.

A seguir, apresenta-se uma descrição de cada um deles, tendo por base a documentação oficial da biblioteca disponível em [27], sendo abordado o seu funcionamento específico, a respetiva justificação para a sua escolha e os parâmetros a ele associados. Para além disso, apresentamos ainda um exemplo simples em código Python de como utilizar na prática estes mecanismos. Após a definição e configuração dos parâmetros, é aplicada, em todos os casos, a função *randomise*, responsável por introduzir o ruído aleatório nos dados, obtendo o nosso valor final anonimizado.

A. Mecanismo de Laplace — `LaplaceBoundedDomain`

Descrição: `LaplaceBoundedDomain` é uma implementação do Mecanismo de Laplace adaptado a um domínio limitado, isto é, o ruído é gerado de forma controlada, garantindo que os valores deturpados permanecem dentro dos limites definidos (`lower` e `upper`). A utilidade deste mecanismo destaca-se em variáveis com intervalos definidos, como a idade.

Justificação da Escolha: Esta escolha foi feita em detrimento do Mecanismo de Laplace padrão, uma vez que é necessário limitar os valores com ruído ao domínio esperado para garantir que continuam a fazer sentido. Desta forma, conseguimos evitar a produção de resultados incoerentes como, por exemplo, idades negativas ou percentagens superiores a cem.

Parâmetros Principais:

- `epsilon` — Parâmetro de privacidade (ϵ)
- `delta` — Parâmetro de privacidade (δ).
- `sensitivity` — Sensibilidade da função.
- `lower/upper` — Limites inferior e superior do domínio.

`Nota:` Embora seja possível atribuir ao parâmetro delta (δ) qualquer valor no intervalo $[0, 1]$, este deve manter-se no valor padrão de 0.0. Isto deve-se ao facto do Mecanismo de Laplace, conforme referido em 2.8, garantir Privacidade Diferencial Pura, o que implica que $\delta = 0$.

Exemplo em Python:

```
from diffprivlib.mechanisms import LaplaceBoundedDomain

laplaceBd = LaplaceBoundedDomain(epsilon=1.0, delta=0.0, sensitivity=10,
                                                    lower=0, upper=100)

valor_original = 75.0
valor_anonimizado = laplaceBd.randomise(valor_original)
```

B. Mecanismo Gaussiano — `GaussianAnalytic`

Descrição: O `GaussianAnalytic` é uma implementação aprimorada do Mecanismo Gaussiano, que utiliza uma calibração analítica conforme proposto por Balle e Wang [5]. Essa abordagem calcula o desvio padrão do ruído de forma mais precisa, garantindo um equilíbrio também ele mais otimizado entre privacidade e utilidade dos dados em análise.

Justificação da Escolha: Essa escolha foi feita em detrimento do Mecanismo Gaussiano padrão, pois a versão selecionada possibilita a adição de um ruído menor para o mesmo nível de privacidade, tornando possível uma melhor utilidade dos dados. Além disso, o mecanismo permite que $\epsilon > 1$, ao contrário do mecanismo padrão, que é limitado a valores entre 0 e 1.

Parâmetros Principais:

- `epsilon` — Parâmetro de privacidade (ϵ).
- `delta` — Parâmetro de privacidade (δ).
- `sensitivity` — Sensibilidade da função protegida.

Exemplo em Python:

```
from diffprivlib.mechanisms import GaussianAnalytic

gaussianA = GaussianAnalytic(epsilon=1.5, delta=1e-5, sensitivity=10)
```

```
valor_original = 75.0
valor_anonimizado = gaussianA.randomise(valor_original)
```

C. Mecanismo Exponencial — `ExponentialCategorical`

Descrição: O `ExponentialCategorical` é uma implementação do Mecanismo Exponencial, adaptada especificamente para variáveis categóricas. Esta versão garante Privacidade Diferencial ao substituir um valor original por outro pertencente à mesma lista de categorias, atribuindo maior probabilidade de escolha aos valores mais "semelhantes" ao valor original.

Para que o mecanismo funcione corretamente, é necessário fornecer uma lista de utilidades, isto é, uma lista que define a proximidade entre os pares de valores. Valores mais próximos devem ter utilidades mais altas, enquanto pares mais distintos recebem utilidades mais baixas.

Justificação da Escolha: Este mecanismo foi o selecionado por ser o mais adequado para variáveis categóricas simples, como cores ou localidades, que surgem com frequência em bases de dados reais. Além disso, permite controlar a aleatoriedade com base na semelhança entre categorias, o que ajuda a preservar a utilidade dos dados mesmo após a anonimização.

Parâmetros Principais:

- `epsilon` — Parâmetro de privacidade (ϵ).
- `utility_list` — Lista de utilidades no formato (dado1, dado2, utilidade)

`Nota:` Na nossa implementação do módulo, os pares com valores iguais como, por exemplo, ("vermelho", "vermelho"), receberam a utilidade máxima (1.0) de modo a aumentar a probabilidade de manter o valor real do mesmo. Já para os outros pares, como não sabíamos ao certo quão parecidos eram, utilizámos o valor 0.5. Este valor representa uma semelhança média entre as categorias, equilibrando dois principais objetivos: introduzir aleatoriedade suficiente para preservar a privacidade e, ao mesmo tempo, manter a utilidade dos dados.

Exemplo em Python:

```
from diffprivlib.mechanisms import ExponentialCategorical

utility_list = [
    ("vermelho", "vermelho", 1.0),
    ("vermelho", "laranja", 0.5),
    ("vermelho", "azul", 0.5),
    ("vermelho", "roxo", 0.5),
    ("vermelho", "amarelo", 0.5),
]

exponencial = ExponentialCategorical(epsilon=1.0, utility_list=utility_list)

valor_original = "vermelho"
valor_anonimizado = exponencial.randomise(valor_original)
```

Conforme analisado acima, os mecanismos são estruturas muito importantes na nossa solução,

apresentando um elevado nível de complexidade devido a todos os parâmetros e cálculos envolvidos para o seu eficaz funcionamento. Estes três mecanismos descritos são aqueles que estão disponíveis aos utilizadores na nossa ferramenta, cabendo agora a cada um desses utilizadores selecionar o que melhor encaixa na sua conjuntura, tendo sempre em atenção as regras de seleção descritas na secção 2.8, acompanhada de uma tabela síntese auxiliar visível em 2.1.

Contudo, esta biblioteca não se limita apenas aos mecanismos, oferecendo também tantas outras funcionalidades relevantes que foram, em certos momentos, úteis ao longo do desenvolvimento deste módulo. Seguem-se alguns desses recursos, acompanhados de uma breve descrição e da forma com que foram utilizados para nos auxiliar no desenvolvimento desta ferramenta:

1) Orçamento de Privacidade: A biblioteca fornece uma classe denominada *Budget Account*, que funciona como um contador de orçamento de privacidade, sendo inicializado com os valores máximos de ϵ (epsilon) e δ (delta) permitidos, representando o total de privacidade que pode ser consumido pelos utilizadores. Dessa forma, possuí funções que nos permitem controlar o uso desse orçamento ao longo dos pedidos sobre os dados. Entre essas destacam-se: `check(`ϵ, δ`)`, que verifica se ainda existe orçamento disponível para o pretendido; `spend(`ϵ, δ`)`, que efetua a subtração do valor indicado; e `total()`, que retorna o montante total de privacidade já utilizado. Este recurso foi bastante utilizado, sendo sujeito a inúmeras alterações ao longo do tempo, devido à evolução que a nossa abordagem ao controlo de orçamentos apresentou. Este controlo passou a ser efetuado por uma tabela da base de dados, sendo essa gestão descrita já na secção seguinte.

2) Exceções e Avisos: Para assegurar que a nossa implementação manteria um nível técnico e um padrão de qualidade elevado, optámos por utilizar alertas da biblioteca ao invés de simples *prints* ou exceções da própria linguagem Python. Estes alertas permitem que informações sobre eventuais problemas que possam surgir durante a execução de uma consulta sejam comunicadas ao utilizador, possibilitando-lhe identificar quais os desafios a enfrentar. De entre as utilizadas podemos destacar, por exemplo, a exceção `BudgetError`, levantada quando se tenta executar uma consulta que ultrapassa o orçamento de privacidade e o aviso `DiffprivlibCompatibilityWarning` que surge quando um parâmetro fornecido não é compatível ou não é reconhecido pela biblioteca.

3) Estatísticas e Sensibilidades: Como mencionado em 2.6, a sensibilidade não é um parâmetro definido pelo utilizador, mas sim algo calculado automaticamente com base nos dados presentes no contexto em questão. Como tal, faltava-nos definir quais os cálculos que deveriam ser realizados para cada estatística ou dado individual, já que este valor varia conforme cada caso. A biblioteca disponibiliza funcionalidades que aplicam a técnica de Privacidade Diferencial a várias métricas. Por isso, analisámos o código-fonte das funções associadas a cada estatística para identificar as sensibilidades específicas utilizadas. Com base nessa análise, complementada com um estudo da biblioteca [26], elaborámos a Tabela 4.6 que fornece os quatro contextos inseridos na ferramenta:

Contexto	Sensibilidade	Descrição
Contagem	1	Cada elemento adicionado ou removido altera o resultado em questão no máximo numa unidade.
Soma	$\max - \min$	Representa a diferença máxima possível na soma caso um elemento seja adicionado ou removido.
Média	$\frac{\max - \min}{n}$	Considera o intervalo dos valores dividido pelo número de elementos (n) assumindo a alteração de um valor.
Valor individual	$\max - \min$	Igual à diferença dos valores possíveis, pois a alteração do valor individual pode variar entre o mínimo e o máximo.

Tabela 4.6: Cálculos para o Parâmetro Sensibilidade

4.6.4 Estrutura da Base de Dados Relacional

Com todas as informações reunidas, iniciámos a construção estruturada de uma base de dados que contemple todos os dados necessários para o correto funcionamento da técnica de Privacidade Diferencial. Durante toda esta fase, procurou-se sempre garantir que a divisão da informação entre as tabelas fosse feita da forma mais organizada, percetível e acessível possível, de modo a facilitar a gestão e compreensão de todo o conjunto de dados envolvidos neste processo.

Ao longo da implementação, foram realizadas inúmeras alterações à estrutura inicial das tabelas, devido aos avanços que foram sendo feitos no projeto e à necessidade de ir inserindo ou removendo variáveis em cada tabela. Essas alterações foram realizadas com o auxílio da ferramenta já anteriormente descrita, o Flyway. Finalmente, alcançámos a planificação final apresentada em 4.8, composta por quatro tabelas principais, além de uma tabela adicional, designada de `flyway_Schema_History`. Esta última é criada automaticamente pelo Flyway e contém informações acerca de todas atualizações efetuadas nas restantes tabelas ao longo do tempo.

Após a definição da estrutura principal da base de dados apresentada na Figura 4.8, surgiu a necessidade de responder a um dos requisitos principais determinado pela Trust Systems: garantir a possibilidade de serem realizadas futuras auditorias aos dados manipulados ao longo do tempo.

Com esse propósito, foi feita uma análise às diversas bibliotecas Python que permitissem a implementação deste controlo nas tabelas do nosso conjunto de dados. Entre as opções disponíveis, constatou-se que a biblioteca `SQLAlchemy-Continuum` era a seleção mais eficaz ao permitir o versionamento automático das tabelas, isto é, o registo constante de qualquer alteração.

A introdução deste mecanismo resultou na criação automática de uma tabela auxiliar, denominada `transaction`, responsável por armazenar um *id* único e a data/hora correspondente a cada operação. Para além disso, cada uma das tabelas principais já registadas passou a ter associada uma outra tabela semelhante, mas contendo agora o registo de todas as alterações identificadas. Essas novas tabelas apresentam duas colunas adicionais em relação às principais: uma chave estrangeira

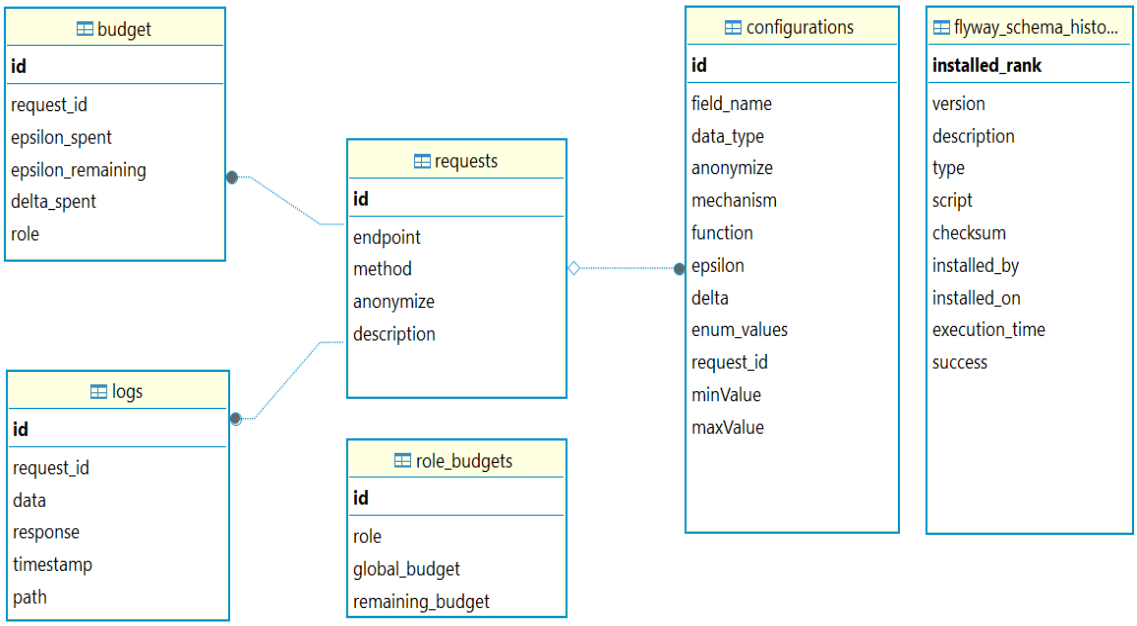

Figura 4.8: Diagrama Entidade Relacionamento (DER) I

que referenciava a transação correspondente e um campo designado operation_type, responsável por indicar o tipo de operação (0 = inserção, 1 = atualização, 2 = eliminação).

Desta forma, a estrutura inicial visualizada na Figura 4.8 foi complementada com novas tabelas auxiliares, responsáveis por manter o histórico das alterações efetuadas em todos os dados. Essas tabelas adicionais ajudaram-nos a cumprir um requisito importante para a empresa em qualquer um dos seus projetos, estando essas tabelas representadas, já a seguir, na Figura 4.9

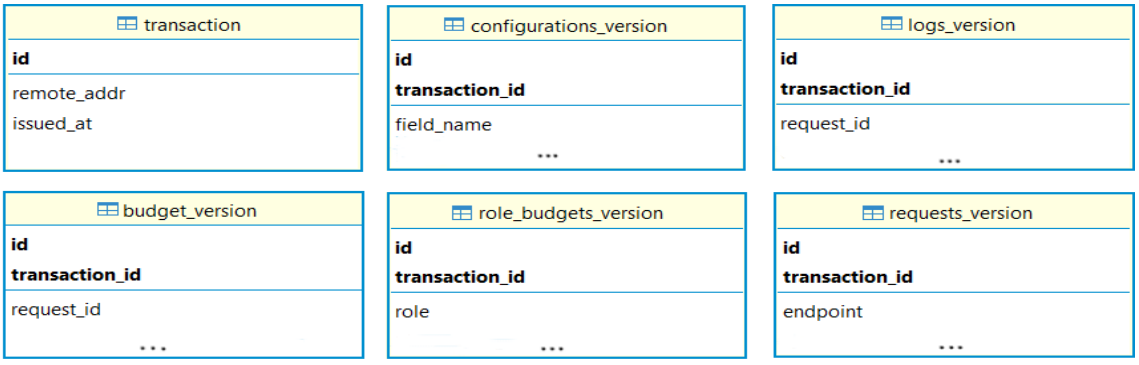

Figura 4.9: Diagrama Entidade Relacionamento (DER) II

Com a junção e consolidação dos Diagramas 4.8 e 4.9, obtemos o esquema final da composição da base de dados. Antes de entendermos todo o fluxo temporal do procedimento desta ferramenta, é preciso perceber de forma detalhada do que se trata cada tabela, para que serve e que dados detêm na sua composição. Desta forma, em seguida, serão apresentadas, de forma individual, as

cinco principais tabelas já mencionadas anteriormente: Tabela 4.7, 4.8, 4.9, 4.10 e 4.11.

Tabela *Requests*

CAMPO	TIPO	DESCRIÇÃO
id	Integer	Identificador único do pedido
endpoint	String	Endpoint da API
method	String	Método HTTP (GET, POST, etc.)
anonymize	Boolean	Indica se o pedido vai ser anonimizado
description	String	Descrição opcional para o pedido

Tabela 4.7: `requests`

A tabela 4.7 representa os pedidos que podem conter informações sensíveis e que, portanto, estão sujeitos à anonimização através da técnica de Privacidade Diferencial. Essa tabela é preenchida pela empresa, que identifica e regista os pedidos que considera sensíveis e, como tal, com dados que apresentam necessidade de proteção. Para esse procedimento, é necessário identificar o endpoint correspondente, o método HTTP, um valor booleano a confirmar a anonimização e, finalmente, quando se achar necessário e relevante, uma descrição relativa ao pedido em questão.

Tabela *Configurations*

CAMPO	TIPO	DESCRIÇÃO
id	Integer	Identificador único da configuração
field_name	String	Nome do campo a ser anonimizado
data_type	String	Tipo de dado (ex: inteiro, float, string)
anonymize	Boolean	Indica se o campo será anonimizado
mechanism	String	Mecanismo de Privacidade Diferencial
function	String	Consulta (média, soma, dado individual)
epsilon	Float	Parâmetro Orçamento de Privacidade ϵ
delta	Float	Parâmetro Delta δ
enum_values	JSON	Lista de Dados categóricos
minValue	Float	Valor Mínimo.
maxValue	Float	Valor Máximo.
request_id	Integer (FK)	Chave estrangeira para `requests`

Tabela 4.8: `configurations`

A tabela 4.8 é uma das mais relevantes desta solução, pois é nela que o utilizador define quais os dados que pretende anonimizar e de que forma o pretende fazer. Para tal, o utilizador tem acesso

aos pedidos listados na tabela `requests` e especifica que dados das respostas desses pedidos sensíveis deverão ser ou não anonimizados. Dessa forma, é necessário preencher informações obrigatórias como o nome do dado, o seu tipo, o mecanismo de Privacidade Diferencial a ser aplicado, a estatística ou valor individual representado na resposta, bem como o valor do orçamento de privacidade (ϵ) a ser consumido para aquele dado e, se for o caso, o valor do parâmetro delta.

Os mecanismos e funções disponíveis na ferramenta são aqueles já previamente descritos nas secções anteriores, sendo, por isso, os únicos valores aceites e válidos na ferramenta. No caso do mecanismo diferencial selecionado ser o Exponencial, utilizado para variáveis categóricas, cabe ao utilizador especificar quais as possíveis categorias correspondentes a esse mesmo campo. Para os restantes mecanismos, é imprescindível que o utilizador mencione quais os valores máximos e mínimos do domínio desse dado em concreto, de modo a garantirmos valores coerentes e de forma a tornar possível o cálculo do parâmetro crítico que é a sensibilidade do pedido.

É importante realçar que a nossa ferramenta não possui acesso direto à base de dados, recebendo apenas as respostas originais provenientes do pedido efetuado. Como tal, cabe ao utilizador fornecer informações precisas e confiáveis, garantindo que os dados sejam corretamente configurados e que a técnica de Privacidade Diferencial seja aplicada da forma mais eficaz possível.

Esta tabela ilustra, de forma clara, o que descrevemos na parte inicial deste Módulo, em 4.6.1, onde, desde a fase de idealização da solução, procurámos torná-la o mais configurável possível, garantindo ao utilizador a liberdade de definir parte dos valores dos parâmetros a que tem acesso.

Tabela *Role_Budgets*

CAMPO	TIPO	DESCRIÇÃO
id	Integer	Identificador único da *role*
role	String	*Role* do utilizador
global_budget	Float	Orçamento total de ϵ permitido
remaining_budget	Float	Orçamento restante disponível

Tabela 4.9: `role_budgets`

A tabela 4.9 tem como objetivo garantir uma gestão adequada do orçamento de privacidade. Ao longo do projeto, foi reforçado que o orçamento de privacidade a ser utilizado pelos utilizadores era um parâmetro crítico, tanto pela sua importância quanto pelo facto de ser um valor limitado.

Inicialmente, considerou-se um orçamento global único, consumido pelos utilizadores à medida que iam sendo realizados pedidos na aplicação. No entanto, de forma a tornar a gestão deste parâmetro mais específica e adequada para diferentes funções dos utilizadores, investigámos algumas abordagens que procurassem repartir este valor. No trabalho realizado [1] por Krishna et al. os investigadores propuseram dividir o orçamento global em diferentes partes, permitindo que diferentes utilizadores tivessem níveis de proteção adaptados às suas necessidades.

A partir dessa abordagem, desenvolvemos a nossa solução, onde dividimos o orçamento global em pequenos orçamentos globais, cada um atribuído a grupos de utilizadores com diferentes *roles* (funções) na empresa, como equipa de administradores, tecnologias ou de recursos humanos. Cada grupo partilha assim do mesmo orçamento, permitindo uma gestão do mesmo mais eficiente.

Assim, a tabela apresenta, para cada *role*, o valor correspondente ao orçamento que lhe é atribuído e também o orçamento restante, tornando possível um controlo elevado deste elemento. De salientar que a função empregue a cada utilizador nada tem a ver com o Módulo de Privacidade Diferencial, mas sim com o contexto derivado da aplicação selecionada para ser intercetada.

Tabela *Budget*

CAMPO	TIPO	DESCRIÇÃO
id	Integer	Identificador único do orçamento
request_id	Integer (FK)	Chave estrangeira para `requests`
epsilon_spent	Float	Valor de ϵ consumido
epsilon_remaining	Float	Valor de ϵ restante
delta_spent	Float	Valor de δ consumido
role	String	*Role* associada ao pedido.

Tabela 4.10: `budget`

A tabela 4.10 tem como objetivo principal possibilitar o controlo do orçamento de privacidade à medida que os pedidos vão sendo executados na aplicação. Ela permite-nos observar, para cada um dos pedidos realizados, o gasto do orçamento de privacidade e do parâmetro de privacidade delta, o valor restante desse mesmo orçamento, bem como a respetiva função do utilizador que realizou o pedido em questão. Cada registo desta tabela permite, assim, monitorizar de forma detalhada onde, como e por quem estes parâmetros estão a ser gastos ao longo do tempo.

Tabela *Logs*

CAMPO	TIPO	DESCRIÇÃO
id	Integer	Identificador único do *Log*
request_id	Integer (FK)	Chave estrangeira para `requests`
data	JSON	Dados de entrada recebidos
response	JSON	Dados retornados após a anonimização
timestamp	Timestamp	Data/hora do pedido
path	String	*Endpoint* da API

Tabela 4.11: `logs`

Como referido no início desta secção, a possibilidade de auditoria e controlo tanto sobre os dados reais como sobre os dados anonimizados, foi identificada como um requisito importante a ter em conta pela Trust Systems. Assim, para além da criação das tabelas de versionamento anteriormente descritas, foi também elaborada a tabela 4.11, acrescentando mais uma camada adicional de monitorização e controlo sobre os resultados que vão sendo obtidos.

Esta tabela representa uma visão comparativa entre o antes e o depois da aplicação da técnica de Privacidade Diferencial, funcionando como um "quadro resumo" final de todo este processo de anonimização. Neste conjunto de dados estão presentes os dados reais, os dados deturpados, o *endpoint* do pedido e, finalmente, a data e hora em que esse pedido foi executado pelo utilizador.

4.6.5 Pedidos Principais Implementados

A nossa implementação apresenta uma estrutura organizada de pastas, classes, funções principais e auxiliares, bem como pedidos que, em conjunto, garantem o funcionamento da nossa solução. De entre todos os pedidos, destacam-se aqueles diretamente relacionados à nossa base de dados, que permitem que toda esta lógica funcione e que consigamos articular e trabalhar de forma eficaz com todo o conjunto de dados que ela detém. Finalmente, dar destaque ao pedido responsável por coordenar todo o fluxo da aplicação desta técnica de anonimização, funcionando como o núcleo central onde ocorre todo o processamento: o nosso Algoritmo de Privacidade Diferencial.

É importante reforçar que todo esse processo é suportado por diversas funções auxiliares que garantem um apoio essencial a todos os restantes pedidos implementados. A seguir, apresentam-se os principais pedidos que compõem esta estrutura, acompanhados de uma breve descrição:

Pedidos Tabela `Requests`	POST /add_request - Adicionar novo *request* GET /get_all_requests - Listar todos os *requests* GET /get_request/{id} - Obter *request* por ID PUT /update_request/{id} - Atualizar *request* DELETE /delete_request/{id} - Apagar *request*
Pedidos Tabela `Configurations`	POST /add_config - Adicionar nova *configuration* GET /get_all_configs - Listar todas as *configurations* GET /get_config/{id} - Obter *configuration* por ID PUT /update_config/{id} - Atualizar *configuration* DELETE /delete_config/{id} - Apagar *configuration*
Pedidos Tabela `Role_Budgets`	POST /add_role_budget - Adicionar novo *role budget* GET /get_all_role_budgets - Listar todos os *role budgets* GET /get_role_budget/{id} - Obter *role budget* por ID PUT /update_role_budget/{id} - Atualizar *role budget* DELETE /delete_role_budget/{id} - Apagar *role budget*
Pedidos Tabela `Budgets`	POST /add_budget - Adicionar novo *budget* GET /get_all_budgets - Listar todos os *budgets* GET /get_budget/{id} - Obter *budget* por ID PUT /update_budget/{id} - Atualizar *budget* DELETE /delete_budget/{id} - Apagar *budget*
Pedidos Tabela `Logs`	POST /add_log - Adicionar novo *log* GET /get_all_logs - Listar todos os *logs* GET /get_log/{id} - Obter *log* por ID DELETE /delete_log/{id} - Apagar *log*

Pedido Principal do Algoritmo DP	POST /apply_dp - Pedido fundamental da nossa ferramenta, contendo toda a lógica por detrás da aplicação técnica de Privacidade Diferencial. Este recebe todos os dados de uma resposta e aplica o seu algoritmo Diferencial, retornando-os devidamente anonimizados.
Pedidos Auxiliares	Aqui foram destacados apenas os pedidos mais relevantes. Contudo, toda a implementação envolve diversas funções auxiliares essenciais para o funcionamento do sistema. Grande parte dessas funções está incluída no algoritmo principal, pedido muito extenso e complexo.

4.6.6 Fluxo e Funcionamento Completo

Com toda a informação analisada e organizada em tabelas, e com todos os pedidos corretamente implementados, o fluxo completo do pedido, desde a sua execução até à aplicação do algoritmo de Privacidade Diferencial e à consequente deturpação dos dados, encontra-se agora estabelecido. De modo a compreendermos o funcionamento detalhado desta solução, em particular o processamento do pedido ao longo do tempo, assim como as ligações entre as diferentes tabelas e as sucessivas validações realizadas, apresentamos, já a seguir, uma descrição sequencial de todo este fluxo:

1. Conforme já referido, o nosso Módulo apenas tem acesso à resposta do pedido executado na aplicação intercetada, não mantendo qualquer tipo de comunicação direta com a base de dados da mesma. Assim, quando o pedido é realizado, o *middleware* já implementado entra em ação até encaminhar esse mesmo pedido ao Módulo de Privacidade Diferencial. É precisamente neste ponto que se inicia o fluxo específico desta componente da solução. Como ilustrado em 4.6, o *middleware* encaminha o pedido, que é recebido pelo *endpoint* `/apply_dp`, juntamente com os potenciais dados a serem deturpados, o pedido original e o método HTTP utilizado. O `/apply_dp` representa o algoritmo principal onde toda a "ação" se vai desenrolar, aplicando toda a lógica necessária aos dados transmitidos;

2. O passo seguinte consiste em verificar se o pedido recebido se encontra listado no conjunto de registos que a tabela `requests` possui. Esta verificação é realizada através da comparação exata do caminho do pedido e do método HTTP utilizado. Caso não seja encontrada correspondência, ou se a coluna *anonymize* estiver desativada, os dados são imediatamente retornados ao *middleware* e, posteriormente, ao utilizador, sem qualquer aplicação de ruído. Por outro lado, se existir uma correspondência, o fluxo prossegue para as etapas seguintes. Esta abordagem permite ignorar pedidos considerados pela empresa como não sensíveis, selecionando apenas aqueles que, potencialmente, necessitam de anonimização;

3. Tendo agora em posse um pedido que é considerado pela empresa como sensível, o sistema consulta a tabela `configurations` para obter todos os parâmetros previamente preenchidos pelo utilizador da nossa ferramenta. O utilizador pode preencher qualquer uma das variáveis conforme considerar mais adequado ao seu contexto, desde que respeite sempre as limitações e regras de preenchimento específicas para cada parâmetro. Caso os valores inseridos não façam sentido ou não cumpram os requisitos minímos obrigatórios, o algoritmo simplesmente aborta a sua execução e apresenta um erro visível ao utilizador;

4. Antes de fazer uso das configurações obtidas, a ferramenta tem de ter a certeza se o utilizador em questão apresenta orçamento de privacidade mínimo para realizar a anonimização. Para isso, é necessário ter acesso ao papel (*role*) do utilizador no contexto da aplicação intercetada e verificar, na tabela `role_budgets`, qual o orçamento restante destinado a essa classe de utilizadores. Como a atribuições de papéis difere e depende da aplicação escolhida, optámos por definir a função do utilizador de forma *hardcoded*. Em seguida, é realizada uma verificação adicional para confirmar se existe orçamento disponível para o algoritmo poder executar as suas operações, isto é, se o orçamento que o utilizador tem ao dispor é superior a zero. Caso se confirme, todo o processo continua normalmente;

5. No decorrer do fluxo, utilizamos o `BudgetAccountant` da biblioteca *diffprivlib* para controlar o consumo do orçamento de privacidade. Inicializamos este objeto com o valor correspondente ao orçamento restante do papel (*role* do utilizador, obtido na tabela `role_budgets`, mencionada já anteriormente. Esse valor acumula o gasto total de privacidade à medida que aplicamos ruído a cada dado, garantindo que não ultrapassamos o limite estabelecido. O uso deste objeto `BudgetAccountant` permite cumprir uma das principais propriedades da técnica Privacidade Diferencial: a Composição (Secção 2.7);

6. A partir deste ponto, inicia-se a aplicação do ruído por meio dos mecanismos de Privacidade Diferencial. O algoritmo consulta o conjunto de configurações disponíveis, preenchendo os parâmetros dos mecanismos com os dados obtidos dessas configurações. Para cada valor a anonimizar, verifica-se se o orçamento de privacidade restante do utilizador é suficiente para aplicar o ruído necessário. Caso haja orçamento disponível, procede-se sempre à aplicação do ruído, caso contrário, o processo é interrompido. Sempre que o orçamento não for suficiente, é lançado o `BudgetError`, indicando o esgotamento do mesmo;

7. Após a aplicação do ruído aos dados selecionados pelo utilizador, calcula-se o valor do orçamento de privacidade e do parâmetro delta consumidos durante a execução. Em seguida, atualiza-se a base de dados: é inserida uma nova linha na tabela `budget`, contendo os valores atualizados desses parâmetros. Também é atualizada a tabela `role_budgets`, ajustando o campo *epsilon_remaining* associado ao papel do utilizador, subtraindo o valor consumido ao valor inicial disponível. Além disso, cria-se uma nova entrada na tabela `logs`, que armazena os dados originais, os dados com ruído e a data em que tudo ocorreu;

8. Por fim, o módulo conclui o processo retornando ao *middleware* previamente implementado o JSON da resposta, que já contém o ruído aplicado aos dados. Dessa forma, o *middleware* guarda a resposta na cache Redis ao mesmo tempo que encaminha essa mesma resposta para o utilizador inicial, garantindo que as informações partilhadas estão devidamente protegidas.

   O Redis desempenha aqui dois papéis muito importantes: além de poder vir a melhorar o desempenho e a eficiência do sistema, permite mitigar um desafio específico da Privacidade Diferencial, conforme discutido na Secção 4.5.1. Um utilizador poderia realizar consultas repetidas até esgotar o orçamento de privacidade, potencialmente combinando informações para atingir dados privados de um indivíduo concreto. Para reduzir este risco, o Redis armazena temporariamente as respostas durante um período de 5 minutos. Caso a mesma consulta seja realizada dentro desse intervalo, a resposta será idêntica à anterior. Embora esta abordagem não elimine completamente esta limitação do orçamento de privacidade, ela diminui significativamente esse risco, mantendo os dados com um maior nível de proteção.

Capítulo 5

Resultados e Avaliação

Neste capítulo, apresentam-se os resultados obtidos após a conclusão da etapa de implementação da solução proposta. Inicialmente, será realizada uma análise ao desempenho de uma aplicação em três cenários distintos: sem plugin, com plugin e com plugin associado a cache. O objetivo passará por avaliar os tempos de resposta em cada configuração, identificando o impacto da introdução desses componentes na melhoria ou degradação do desempenho geral. Para além disso, serão realizadas algumas análises referentes ao Módulo de Privacidade Diferencial, nas quais variados valores e contextos serão testados, permitindo uma correta interpretação dos resultados obtidos.

5.1 Análise I: Middleware Personalizado e Cache Redis

Tal como descrito no capítulo anterior, para cumprirmos com todos os objetivos inicialmente delineados, foi criado um *middleware* personalizado, isto é, um novo *plugin* em GO. A introdução deste novo componente na arquitetura da solução, uma ação inevitável face aos requisitos do projeto, suscitou algumas preocupações quanto a um eventual aumento do tempo de processamento de cada pedido, uma vez que tal poderia vir a comprometer a experiência do utilizador.

Assim, de forma a mitigar esse impacto, recorremos à utilização de uma cache temporária com recurso ao Redis. Esta solução permitiu armazenar as respostas dos pedidos durante um intervalo de tempo pré-definido, evitando que os pedidos repetidos nesse período precisassem de percorrer o fluxo de processamento completo, passando a ser recuperados diretamente da cache.

O foco desta secção é, portanto, avaliar o impacto da inclusão do *plugin* e da cache no tempo de processamento de pedidos numa aplicação real. Para esse fim, foi selecionado o CRM - REACH, um sistema de *Customer Relationship Management* (CRM) utilizado pela empresa. Este sistema foi desenvolvido para suportar a atividade diária de diferentes perfis de utilizadores, assegurando uma gestão eficaz de informações pessoais e sensíveis de mais de dois milhões de indivíduos.

A Trust Systems, empresa que participou no desenvolvimento e implementação do sistema, possui acesso autorizado ao CRM. Devido ao elevado volume e diversidade de dados presentes, este representa um caso de estudo muito relevante, tornando-o uma escolha adequada para a análise e permitindo validar os resultados de forma mais consistente e rigorosa.

Definida a aplicação e efetuada toda a preparação dos passos necessários para a integração do *plugin*, procedeu-se com a avaliação do seu desempenho. O objetivo passou por comparar os tempos de resposta de um mesmo pedido, escolhido de forma arbitrária, em três cenários distintos:

1. Sem Plugin - Pedido diretamente para a aplicação, sem qualquer intervenção;

2. Com Plugin - Pedido que passa pelo *middleware* personalizado;

3. Com Plugin e Cache - Pedido que passa pelo *middleware*, mas utiliza cache temporária.

Para cada um desses cenários definidos, foram conduzidos dois tipos distintos de testes, nos quais foram calculadas diversas métricas específicas para cada um deles. Todos esses testes foram realizados com recurso à ferramenta *Postman*, sendo descritos e apresentados já a seguir:

Teste A

Execução de forma conjunta e simultânea de um mesmo pedido em conjuntos de 20, 40, 60, 80 e 100 repetições.

Teste B

Execução concorrente do mesmo pedido em grupos de 5, 10, 15 e 20 utilizadores, durante um intervalo de tempo de 1 minuto.

Para realizarmos uma interpretação correta dos resultados, procurámos identificar os valores ideais correspondentes ao tempo de processamento de resposta de uma API. Com base no artigo [6], compreendemos que as expectativas de tempo podem variar de acordo com o tipo de API, com o setor de atuação e com o contexto específico no momento da análise. No entanto, dada a importância de estabelecer pontos de referência claros para uma avaliação eficaz, nesse trabalho elaborou-se uma tabela de referência que servirá de guia para a nossa análise. A Tabela 5.1, obtida de [6], organiza os tempos em diferentes níveis de classificação e encontra-se apresentada abaixo:

CLASSIFICAÇÃO	TEMPO (MS)	EXPERIÊNCIA DO UTILIZADOR
Instantâneo	< 100 ms	Resposta imediata, praticamente impercetível
Excelente	100 – 300 ms	Muito rápido, quase impercetível
Muito bom	300 – 500 ms	Levemente percetível, sem impacto significativo
Bom	500 – 800 ms	Percetível, mas aceitável
Razoável	800 – 1000 ms	Ligeiramente lento, começa a ser notado
Lento	1000 – 2000 ms	Notável, pode afetar a experiência
Inaceitável	> 2000 ms	Muito lento, altamente percetível e frustrante

Tabela 5.1: Categorias de tempos de resposta de APIs e Perceção do Utilizador

Além disso, de modo a complementar as futuras análises, calculou-se a diferença percentual da latência média entre os cenários *Sem Plugin* e *Com Plugin* e entre os cenários *Com Plugin* e *Com Plugin + Cache*. Essa métrica indica, em termos relativos, quanto o tempo de resposta mudou entre dois cenários: um valor positivo significa aumento da latência, enquanto um valor negativo indica a redução da mesma. O valor da diferença percentual foi obtido através da seguinte fórmula:

$$\textbf{Diferença (\%)} = \frac{\text{Latência Cenário Comparado} - \text{Latência Cenário Base}}{\text{Latência Cenário Base}} \times 100$$

Esta comparação permite avaliar quantitativamente o impacto do *plugin* sobre o desempenho do sistema, bem como a eficácia da cache na redução do tempo de resposta. Os resultados são apresentados, a seguir, através de tabelas localizadas na secção correspondente a cada um dos testes.

5.1.1 Teste A

O Teste A consistiu na execução simultânea de um mesmo pedido em grupos de 20, 40, 60, 80 e 100 repetições. Esta abordagem foi escolhida por refletir situações reais de uso intenso do sistema, em que múltiplos pedidos podem ser efetuados num curto intervalo de tempo. Assim, conseguimos avaliar a capacidade do sistema em lidar com cargas elevadas e identificar eventuais alterações anormais do seu comportamento. Para esta análise, foram consideradas duas métricas:

- **Duração Total:** Tempo necessário para concluir todos os pedidos de cada conjunto;
- **Latência Média:** Tempo médio de resposta por pedido.

Após a execução dos testes e o cálculo das métricas nos diferentes cenários, foi elaborada a Tabela 5.2, acompanhada dos gráficos da Figura 5.1, de forma a apresentar os resultados de forma mais visual. Estes elementos permitem uma análise de todos os dados, conforme apresentado a seguir:

Nº Pedidos	Sem Plugin		Com Plugin		Com Plugin + Redis	
	Duração	Latência	Duração	Latência	Duração	Latência
20	4 s 558 ms	192 ms	7 s 319 ms	328 ms	3 s 501 ms	103 ms
40	7 s 918 ms	178 ms	13 s 173 ms	305 ms	4 s 420 ms	85 ms
60	11 s 487 ms	173 ms	19 s 543 ms	306 ms	5 s 992 ms	78 ms
80	14 s 935 ms	172 ms	28 s 456 ms	334 ms	8 s 317 ms	79 ms
100	18 s 908 ms	171 ms	37 s 879 ms	308 ms	9 s 862 ms	78 ms

Tabela 5.2: Resultados - Teste A

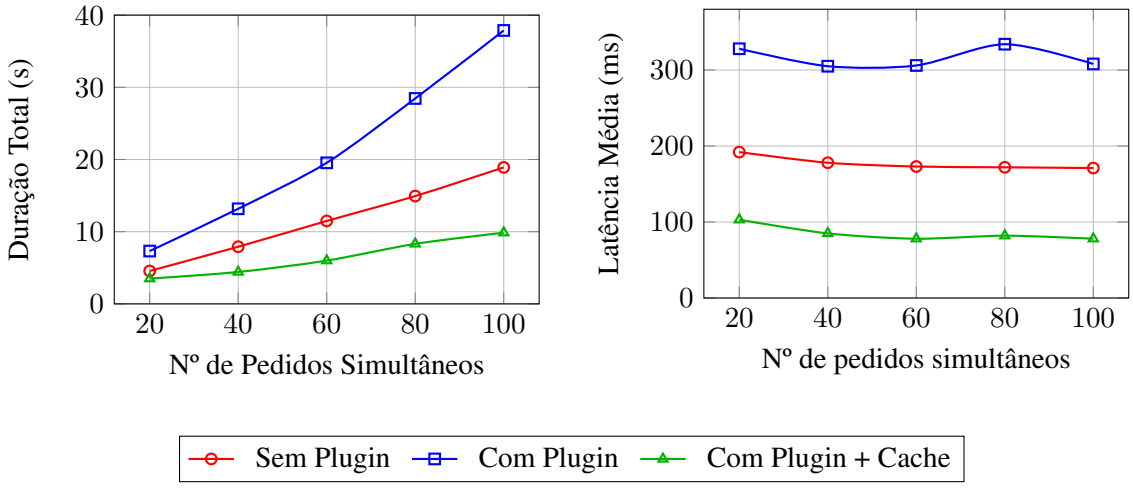

Figura 5.1: Comparação entre Duração Total e Latência Média por cenário

Com os dados devidamente calculados, podemos avançar para o cálculo da métrica já introduzida inicialmente: a diferença percentual entre dois cenários distintos relativamente à latência média. As Tabelas 5.3 e 5.4 apresentam todos os resultados obtidos, permitindo uma análise comparativa:

Nº Pedidos	Diferença (%)
20	((7.319-4.558) / 4.558) × 100 ≈ 60.6%
40	((13.173-7.918) / 7.918) × 100 ≈ 66.4%
60	((19.543-11.487) / 11.487) × 100 ≈ 70.1%
80	((28.456-14.935) / 14.935) × 100 ≈ 90.5%
100	((37.879-18.908) / 18.908) × 100 ≈ 100.3%

Tabela 5.3: A - *Sem Plugin* / *Com Plugin*

Nº Pedidos	Diferença (%)
20	((3.501-7.319) / 7.319) × 100 ≈ -52.1%
40	((4.420-13.173) / 13.173) × 100 ≈ -66.5%
60	((5.992-19.543) / 19.543) × 100 ≈ -69.3%
80	((8.317-28.456) / 28.456) × 100 ≈ -70.8%
100	((9.862-37.879) / 37.879) × 100 ≈ -73.9%

Tabela 5.4: A - *Com Plugin*/*Com Plugin* + *Cache*

Análise

Os resultados obtidos evidenciam diferenças significativas entre os três cenários analisados:

⇒ **SEM PLUGIN**: A latência média mantém-se praticamente estável, apresentando resultados a variar entre 171 ms (100 pedidos) e 192 ms (20 pedidos), enquanto a duração total aumenta linearmente com o número de pedidos realizados, como pode ser observado no gráfico da Figura 5.1. Os valores da latência média refletem um eficaz desempenho da aplicação, encontrando-se os seus valores dentro da categoria "Excelente" da tabela de referência 5.1.

⇒ **COM PLUGIN:** A introdução do *plugin* aumenta as métricas calculadas de forma nítida. Os valores da latência média passam a variar entre os 305 ms e os 334 ms, enquanto a duração total de execução passa a atingir o valor máximo de 37.879 s. No gráfico de duração total da Figura 5.1, a linha azul apresenta um crescimento acentuado, indicando que o *plugin* adiciona

processamento extra aos pedidos, o que gera um aumento no tempo de resposta (*overhead*). Esse aumento decorre do processamento adicional realizado pelo Módulo de Privacidade Diferencial, para o qual o plugin redireciona os pedidos. Contudo, apesar deste aumento para mais do dobro, as latências permanecem abaixo de 500 ms, encontrando-se na categoria "Muito Bom".

⇒ **COM PLUGIN + REDIS:** A introdução do mecanismo de cache demonstrou grande eficácia, reduzindo drasticamente a latência média, que varia entre 78 ms (100 pedidos) e 103 ms (20 pedidos), e a duração total, que se situa entre 3.501 s e 9.862 s, respetivamente. Como esperado, ao efetuar o mesmo pedido com a cache ativa, mesmo com o aumento do número de pedidos a serem executados, os valores permanecem muito baixos, uma vez que as respostas são diretamente da cache temporária. Nos gráficos em 5.1, a linha verde destaca-se por permanecer sempre abaixo das demais, demonstrando claramente a eficiência da cache. Deste modo, estes resultados observados enquadram-se entre categorias "Instantâneo" e "Excelente" da tabela de referência 5.1.

Conclusões

Observados os resultados e realizada uma análise aos mesmos, chegamos às seguintes constatações:

1. O *plugin*, embora aumente a latência média entre 60.6% (20 pedidos) e 100.3% (100 pedidos), mantém o desempenho dentro de limites aceitáveis, confirmando a sua viabilidade.
2. A utilização da cache permite reduzir drasticamente a latência, com diferenças percentuais entre -52,1% (20 pedidos) e -73,9% (100 pedidos) em relação ao cenário apenas Com *Plugin*, evidenciando a sua importância para a otimização do desempenho da aplicação.
3. A latência em *Plugin* + Cache é até inferior a Sem *Plugin*, demonstrando que a cache não apenas compensa o *overhead* do *Plugin*, como também melhora o desempenho da aplicação
4. O teste confirma que a arquitetura proposta é capaz de lidar com o aumento da carga de forma eficiente, garantindo respostas rápidas mesmo em cenários de elevada sobrecarga.

5.1.2 Teste B

O Teste B consistiu na execução de um mesmo pedido em grupos de 5, 10, 15 e 20 utilizadores, durante um intervalo de tempo de um minuto. Este foi mais um dos testes considerados particularmente relevantes, uma vez que reproduz situações de elevada concorrência, em que diferentes utilizadores acedem simultaneamente ao mesmo pedido do sistema, situação bastante comum com aplicações em produção. Assim, para esta análise, foram consideradas quatro diferentes métricas:

- **Número de Pedidos:** Total de pedidos executados durante o intervalo de tempo definido;
- **Pedidos por Segundo:** Total de pedidos executados num segundo;
- **Latência Média:** Tempo médio de resposta por pedido;
- **Percentil 90:** Valor abaixo do qual se encontram 90% das respostas.

Após a execução dos testes e o cálculo das métricas nos diferentes cenários, foi elaborada a Tabela 5.5, acompanhada dos gráficos da Figura 5.2, de forma a representar os resultados de modo mais visual. Estes dados permitem uma análise de todos os resultados, conforme apresentado a seguir:

Users	Sem Plugin				Com Plugin				Com Plugin + Cache			
	Nº Pedidos	Ped./s	Latência	P90	Nº Pedidos	Ped./s	Latência	P90	Nº Pedidos	Ped./s	Latência	P90
5	239	3.48	180 ms	199 ms	217	3.27	313 ms	367 ms	265	3.87	84 ms	72 ms
10	472	6.88	184 ms	208 ms	422	6.24	339 ms	434 ms	525	7.68	70 ms	73 ms
15	683	9.94	188 ms	211 ms	656	9.57	295 ms	394 ms	734	10.59	82 ms	85 ms
20	889	13.10	189 ms	216 ms	852	12.22	272 ms	329 ms	920	13.72	87 ms	94 ms

Tabela 5.5: Resultados - Teste B

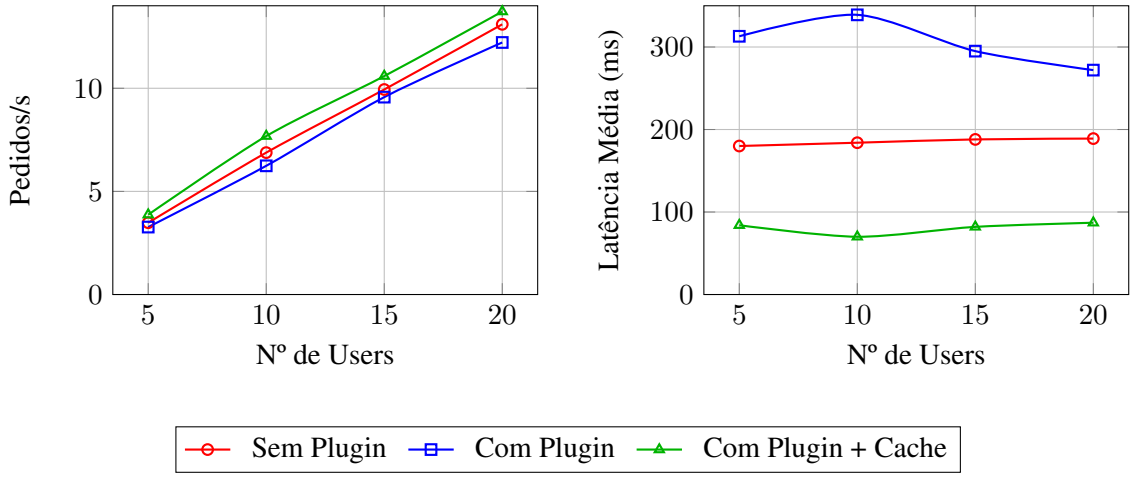

Figura 5.2: Comparação entre Pedidos/s e Latência Média por cenário

Com os dados devidamente calculados, podemos avançar para o cálculo da métrica já introduzida inicialmente: a diferença percentual entre dois cenários distintos relativamente à latência média. As Tabelas 5.3 e 5.4 apresentam todos os resultados obtidos, permitindo uma análise comparativa:

Users	Diferença (%)
5	((313-180) / 180) × 100 ≈ 73.9%
10	((339-184) / 184) × 100 ≈ 84.2%
15	((295-188) / 188) × 100 ≈ 57.4%
20	((272-189) / 189) × 100 ≈ 43.9%

Tabela 5.6: B - *Sem Plugin* / *Com Plugin*

Users	Diferença (%)
5	((84-313) / 313) × 100 ≈ -73.2%
10	((70-339) / 339) × 100 ≈ -79.3%
15	((82-295) / 295) × 100 ≈ -72.2%
20	((87-272) / 272) × 100 ≈ -68.0%

Tabela 5.7: B- *Com Plugin* / *Com Plugin + Cache*

Análise

Os resultados obtidos evidenciam diferenças significativas entre os três cenários analisados:

⇒ **SEM PLUGIN**: A latência média manteve-se relativamente estável, estando os valores situados entre 180 ms (5 utilizadores) e 189 ms (20 utilizadores), com o P90 ligeiramente superior, atingindo os 216 ms. Quanto à taxa de pedidos por segundo, este dado aumentou de forma quase linear, de 3.48 Ped./s até 13.10 Ped./s. Nos gráficos da Figura 5.5, a linha vermelha cresce de forma linear para o número de Ped./s, permanecendo estável na latência, confirmando o bom desempenho do sistema com valores a enquadrarem-se na categoria "Excelente" da tabela 5.1.

⇒ **COM PLUGIN:** Com a introduçao do *plugin*, verificou-se um aumento generalizado da latência média, com valores situados entre 272 ms e 339 ms, e com um P90 a atingir os 434 ms. Ao mesmo tempo, o número de pedidos por segundo registou uma ligeira diminuição em todos os casos, encontrando-se entre 3,27 Ped./s e 12,22 Ped./s, resultados derivados do custo adicional causado pelo processamento do Módulo de Privacidade Diferencial. No entanto, os valores da latência mantêm-se abaixo dos 500 ms, estando situadas na categoria "Muito Bom" e não comprometendo a experiência do utilizador. Nos gráficos da Figura 5.2, a linha azul destaca-se acima da vermelha no gráfico da latência, demonstrando o aumento do tempo de resposta, e ligeiramente abaixo, no gráfico do número de pedidos efetuados, ilustrando o superficial decréscimo deste número.

⇒ **COM PLUGIN + REDIS:** Com a utilização da cache em conjunto com o *plugin*, os resultados revelaram uma melhoria significativa no desempenho. A latência média desceu para valores entre os 70 ms e 87 ms, com o P90 a atingir o valor de 94 ms. Além disso, o número de pedidos aumentou, atingindo 13,72 Ped./s com 20 utilizadores, ultrapassando inclusive o cenário base Sem *Plugin*.A análise da Figura 5.2 reforça esta conclusão, evidenciando que o cenário com cache apresenta a maior taxa de pedidos por segundo em comparação com os restantes. A linha verde destaca-se pela sua estabilidade e por se manter consistentemente em valores de latência significativamente inferiores, inserindo-se, praticamente em todos os valores, na categoria "Instantâneo".

Conclusões

1. O *plugin* aumenta a latência média entre aproximadamente 43,9% e 84,2% em comparação com o cenário Sem *Plugin*, mantendo, no entanto, o desempenho dentro de limites aceitáveis.

2. A utilização da cache reduz drasticamente a latência, com diferenças percentuais entre -68% (20 utilizadores) e -79.3% (10 utilizadores) em relação ao cenário apenas com *plugin*, evidenciando, mais uma vez, a sua relevância na otimização do desempenho da aplicação.

3. A combinação *Plugin + Cache* apresenta latências médias inferiores às do cenário Sem *Plugin*, comprovando que a cache não só compensa o *overhead* do *plugin*, como também melhora globalmente o desempenho do sistema.

4. O teste B, tal como no teste A, confirma que a arquitetura suporta eficientemente aumentos de carga, garantindo respostas rápidas mesmo em situações de elevada concorrência.

5.2 Análise II: Módulo de Privacidade Diferencial

Após a avaliação do desempenho de uma determinada aplicação com o *middleware* personalizado e do impacto da cache no tempo de resposta da mesma, tornou-se essencial analisar a eficácia do componente central da nossa solução implementada: o Módulo de Privacidade Diferencial.

Para tal, foram estruturados alguns testes que procuraram avaliar o módulo sob diferentes perspetivas. De forma simplificada, esta avaliação foi organizada em duas vertentes principais:

- **Anonimização de Dados Numéricos:** Avaliou-se o impacto da adição de ruído nos dados sobre a utilidade de uma estatística agregada, permitindo medir o equilíbrio entre privacidade e precisão. Neste caso, foram utilizados os Mecanismos de Laplace e Gaussiano.
- **Anonimização de Dados Categóricos:** Avaliou-se a aplicação do Mecanismo Exponencial em dados categóricos, verificando se a utilidade das respostas a um determinado pedido de estatística agregada era preservada após a anonimização por parte do módulo implementado.

Para esta análise, foi necessária a utilização de uma aplicação capaz de retornar estatísticas relativas a um determinado grupo de indivíduos. Como a Trust Systems não dispõe de um sistema que nos forneça essas métricas para determinados dados, foi desenvolvida uma pequena aplicação com essa funcionalidade, passando a ser essa a ferramenta intercetada pelo nosso *middleware*. Além disso, para garantir a obtenção de resultados distintos e ser possível realizar uma análise estatística credível e adequada, a cache (Redis) do sistema foi desativada durante toda essa análise.

Após a criação da aplicação, tornou-se necessário selecionar cuidadosamente um conjunto de dados representativo, de modo a garantir que os testes se aproximassem o mais possível de cenários reais, ou seja, que incluíssem informações pessoais e sensíveis relativas a indivíduos.

Durante a exploração de repositórios de dados públicos, recorreu-se à plataforma Kaggle, onde foi possível encontrar um *dataset* relativamente simples que se enquadrava nos critérios previamente estabelecidos. O conjunto de dados, denominado *"Employee Data"* [35], contém informações pessoais de mais de 10.000 funcionários de uma empresa e cumpre com os dois critérios principais procurados: possuir informações críticas e uma grande quantidade de dados.

Para a análise proposta, foram removidas colunas consideradas irrelevantes para a nossa abordagem, resultando no conjunto final apresentado na Tabela 5.8 que inclui algumas das entradas:

NOME	IDADE	GÉNERO	FUNÇÃO	CIDADE	SALÁRIO
John Mayes	56	Male	Sales	Seattle	195000
Carlos Wille	21	Male	Engineering	New York	35000
Michael Bryant	30	Male	Finance	New York	75000
...	...	...	...	...	...

Tabela 5.8: Amostra do *dataset: "Employee Data"* [35]

5.2.1 Teste A: Anonimização de Dados Numéricos – Laplace vs Gaussiano

Objetivo: Avaliar o *trade-off* entre a precisão e o nível de privacidade oferecido por dois mecanismos: Laplace e Gaussiano. Este teste compara o erro introduzido pelos mecanismos para um determinado pedido, testando e analisando o impacto de diferentes valores de ϵ no resultado.

Metodologia

- Foi definido um pedido para retornar a estatística que representa a média salarial dos funcionários representados pelo conjunto de dados selecionado [35] (Valor Real = 115381.500).
- A sensibilidade da "média" é calculada como a diferença entre o salário máximo e o salário mínimo, dividida pelo total de funcionários. Neste caso, temos: (215000 - 25000) / 10.000 = 19.
- Os mecanismos selecionados para este tipo de teste foram o Laplace e o Gaussiano, pois ambos são os mais adequados para lidar com dados numéricos. No caso do Mecanismo Gaussiano, foi utilizado um valor padrão para o parâmetro δ, definido com um valor igual a $\delta = 1 \times 10^{-3}$.
- Foram testados cinco diferentes valores de ϵ para cada um dos dois mecanismos escolhidos:

- Para cada um dos mecanismos selecionados, o mesmo pedido foi executado cerca de 50 vezes.
- Para avaliar a utilidade e precisão dos dados em questão, foi calculada a métrica Erro Percentual Absoluto Médio (Mean Absolute Percentage Error) para cada conjunto de 50 resultados. Essa métrica permite-nos deduzir conclusões muito importantes, sendo obtida pela seguinte fórmula:

$$\text{MAPE} = \frac{1}{N} \sum_{i=1}^{N} \frac{|\text{Valor Real}_i - \text{Valor Anonimizado}_i|}{\text{Valor Real}_i} \times 100\%, \quad \text{onde } N = 50$$

O Erro Percentual Absoluto Médio (MAPE) [17] representa a média das diferenças absolutas entre os valores reais e os valores anonimizados, em relação ao valor real e expressas em percentagem, sendo sempre um valor não negativo. Quanto maior o MAPE, maior é a distância entre os valores anonimizados e os valores reais, ou seja, maior a distorção introduzida pelo Módulo de Privacidade Diferencial. Por outro lado, quanto menor o MAPE, menor é a deturpação.

Resultados e Análise

Com base nos valores calculados para todos os parâmetros que se pretendia testar, obtiveram-se as tabelas mencionadas em 5.9. Ambas apresentam a média dos 50 valores anonimizados e o valor do MAPE, para todas as variações do parâmetro ϵ e para cada um dos mecanismos já abordados. O valor real da média salarial é 115381.500, e as tabelas correspondentes são apresentadas a seguir:

Mecanismo Laplace			**Mecanismo Gaussiano**		
ϵ	Média	MAPE (%)	ϵ	Média	MAPE (%)
0.01	115788.58	1.602	0.01	115980.04	1.110
0.1	115436.74	0.139	0.1	115329.40	0.220
0.5	115396.30	0.029	0.5	115365.65	0.050
1.0	115380.87	0.017	1.0	115381.87	0.030
5.0	115381.20	0.003	5.0	115381.44	0.010

Tabela 5.9: Resultados Teste B: Médias Anonimizadas e MAPE (%)

A análise das tabelas em 5.9 permite extrair conclusões essenciais:

- Os resultados mostram que, para ambos os mecanismos, o MAPE diminui à medida que ϵ aumenta, o que era esperado uma vez que, um maior orçamento de privacidade permite adicionar menos ruído, resultando em valores mais próximos do real. Para $\epsilon = 0.01$, o MAPE é superior a 1%, indicando uma perda de utilidade. Já para $\epsilon = 5.0$, o MAPE é inferior a 0.01%, mostrando uma precisão muito elevada, mas com um custo também elevado em termos de privacidade.

- Comparando ambos os mecanismos mencionados, verifica-se que o Mecanismo Gaussiano tende a apresentar um MAPE ligeiramente inferior para valores muito baixos de ϵ (0.01, por exemplo), enquanto o Mecanismo de Laplace oferece melhor precisão para valores intermédios e até mesmo para os valores mais altos. Tudo isto está de acordo com as características teóricas de cada mecanismo (descritas na Secção 2.8): Gaussiano mais eficiente em contextos de alta privacidade (ϵ pequeno), enquanto Laplace é mais preciso quando se pode gastar mais orçamento.

- Para complementar a análise, a Figura 5.3 apresenta a dispersão das médias dos valores anonimizados em relação ao valor real (representado pela linha horizontal tracejada) para cada valor de ϵ e para ambos os mecanismos. Cada ponto corresponde à média anonimizada obtida para cada ϵ. Observa-se claramente que a distância entre os valores anonimizados e o valor real diminui à medida que ϵ aumenta e que ambos os mecanismos apresentam comportamentos semelhantes.

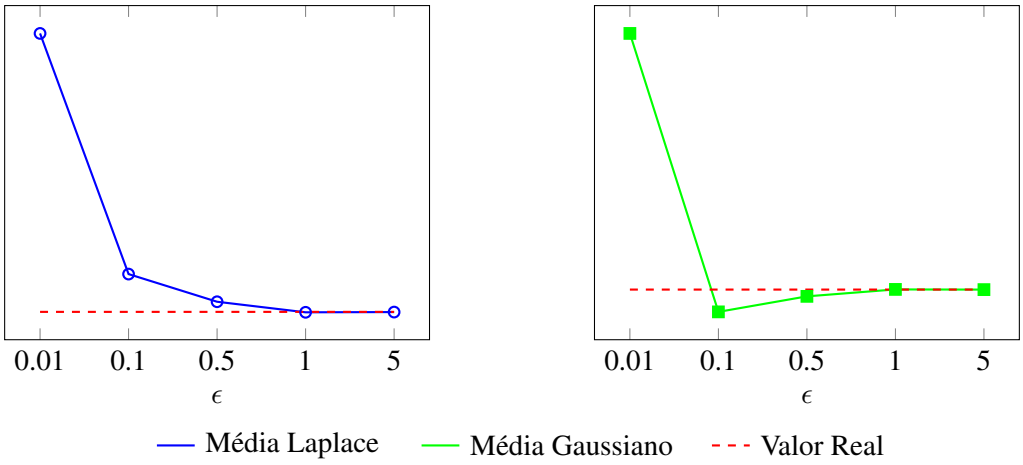

Figura 5.3: Dispersão Valor Real vs Médias Anonimizadas

5.2.2 Teste B: Anonimização de Dados Categóricos – Exponencial

Objetivo: Avaliar a eficácia e o comportamento do Mecanismo Exponencial na proteção de dados categóricos considerados como sensíveis. O foco deste teste consiste em verificar se o mecanismo consegue preservar a utilidade dos dados, ou seja, se tende a manter o seu valor real, ao mesmo tempo que introduz aleatoriedade suficiente para garantir a privacidade dos valores em questão.

Metodologia

- Foi definido um pedido para identificar a função mais comum entre os trabalhadores presentes no conjunto de dados selecionado [35]. Nesse contexto, a função predominante é *"Product"*.
- As opções existentes são: ["*Engineering*", "*Sales*", "*Finance*", "*HR*", "*Marketing*", "*Product*"].
- Como explicado anteriormente na Secção 4.6.3, este mecanismo requer a definição de uma lista de utilidades (*utility list*). Para este teste, essa lista foi configurada de acordo com a semelhança existente entre determinadas categorias, agrupadas em três grandes blocos organizacionais:
  - **Grupo A:** Desenvolvimento de Soluções ("*Product*" + "*Engineering*")
  - **Grupo B:** Expansão de Mercado ("*Marketing*" + "*Sales*")
  - **Grupo C:** Infraestrutura Organizacional ("*Finance*" + "*HR*")

  Dessa forma, as utilidades foram atribuídas aos pares da seguinte forma:
  - ("*Product*" - "*Product*") → Valor real → Utilidade = 1.0
  - ("*Product*" - "*Engineering*") → Mesmo grupo (elevada semelhança) → Utilidade = 0.6
  - ("*Product*" - *Elemento do Grupo B*) → Ligação média entre Grupos → Utilidade = 0.4
  - ("*Product*" *Elemento do Grupo C*) → Ligação fraca entre Grupos → Utilidade = 0.2

- Para avaliar o impacto do orçamento no comportamento do Mecanismo Exponencial, o pedido que retorna a função mais comum ("*Product*") foi executado para cinco valores distintos de ϵ:

- Para cada um dos diferentes valores do orçamento de privacidade em análise, o mesmo pedido foi executado 100 vezes, e no final registou-se a frequência de cada categoria em cada caso.

Resultados e Análise

Os resultados obtidos neste teste estão sintetizados na Tabela 5.10, onde é apresentada a frequência que cada uma das seis categorias apresentou para o respetivo valor de ϵ, ilustrando de forma clara o efeito desse parâmetro no *trade-off* existente entre privacidade e utilidade dos dados.

Categoria	**Frequência (%)**				
	$\epsilon = 0.01$	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1$	$\epsilon = 5$
Product	19	22	24	26	43
Enginnering	20	18	18	21	24
Marketing	18	16	17	14	10
Sales	18	17	16	16	13
Finance	15	14	13	12	6
HR	10	15	12	11	4

Tabela 5.10: Frequência das Categorias para diferentes valores de ϵ

A análise aos resultados permite extrair pontos fundamentais acerca do Mecanismo Exponencial:

- A influência do orçamento de privacidade é evidente e decisiva. Para um valor de $\epsilon = 0.01$ (alta privacidade nos dados), a distribuição dos resultados é quase uniforme. A categoria original *"Product"* aparece apenas 19% das vezes, uma frequência apenas ligeiramente superior à das restantes categorias. Neste cenário em questão, a privacidade é praticamente máxima, uma vez que um adversário ganha muito pouca informação ao observar o valor resultante. Contudo, a utilidade do resultado também é muito baixa, já que o valor raramente é preservado.

- À medida que o valor de ϵ aumenta, a probabilidade do valor original ser selecionado aumenta gradualmente. Para $\epsilon = 0.1$, *"Product"* aparece 22% das vezes; para $\epsilon = 0.5$, aparece 24%; e para $\epsilon = 1.0$, 26%. Nestes valores mais intermédios, estabelece-se um equilíbrio: a utilidade dos dados é melhorada, mas mantém-se um certo grau de incerteza, isto é, de privacidade, com as restantes categorias também a aparecerem com uma frequência muito relevante.

- Para um valor de $\epsilon = 5.0$ (baixa privacidade dos dados), a categoria principal *"Product"* é selecionada em 43% das execuções, enquanto a frequência da maioria das restantes categorias cai para valores mínimos a rondar entre os 4% e 10%. Neste caso, a utilidade é muito alta, mas a privacidade é praticamente comprometida, uma vez que um adversário pode inferir com alta confiança qual era o valor original ao observar os resultados das 100 execuções realizadas.

- A lista de utilidade baseada em grupos semânticos mostrou-se eficaz para preservar a coerência dos dados. Os resultados indicam que, mesmo com $\epsilon = 0.01$, as categorias pertencentes ao mesmo grupo do valor real aparecem com frequências ligeiramente superiores. À medida que o ϵ aumenta (por exemplo, $\epsilon = 5$), as categorias do Grupo A (maior utilidade) dominam os resultados, enquanto as pertencentes a grupos mais distantes tornam-se residuais. Demonstra-se assim a preservação das relações semânticas, garantindo um funcionamento eficaz do mecanismo.

Capítulo 6

Conclusão

6.1 Síntese do Projeto Realizado

Este trabalho teve como objetivo explorar, implementar e avaliar a aplicação da técnica de Privacidade Diferencial, em parceria com a Trust Systems. O projeto foi concretizado com sucesso, resultando no desenvolvimento de uma solução prática e eficaz que permite proteger dados considerados sensíveis de respostas de aplicações externas, garantindo a privacidade dos indivíduos sem comprometer de forma significativa a utilidade dos dados para futuras análises estatísticas.

Ao longo destes meses de trabalho, foram realizadas diversas etapas, iniciando-se com a investigação de toda a informação teórica relacionada à Privacidade Diferencial, incluindo os seus mecanismos, propriedades, bem como os desafios e limitações que apresenta. Seguiu-se a análise do trabalho relacionado, avaliando diversas ferramentas e bibliotecas existentes, culminando na seleção da biblioteca *diffprivlib* para inclusão no nosso Módulo de Privacidade Diferencial.

Seguiu-se a fase de implementação da solução proposta, que resultou no desenvolvimento de dois componentes distintos: o middleware personalizado, criado como um plugin para o Traefik em linguagem Go, responsável por intercetar, processar e redirecionar pedidos e respostas; e o Módulo de Privacidade Diferencial, desenvolvido como um microserviço em Python com FastAPI, que aplica a técnica de acordo com diversas configurações definidas pelo utilizador e que gere vários orçamentos de privacidade globais baseados nas funções (*roles*) de cada utilizador.

Finalmente, foram realizados testes que exploraram e avaliaram o impacto da introdução do plugin e da cache no desempenho global do sistema. Na fase final concluíu-se que, embora o plugin aumente o tempo de resposta de forma inevitável, a introdução da cache compensa essa adição, resultando num desempenho muito bom. Também foram realizados testes no módulo implementado, confirmando a eficácia dos mecanismos e revelando o impacto do parâmetro orçamento de privacidade no *trade-off* existente entre a utilidade e a privacidade dos dados em análise.

Em suma, este projeto contribuiu com uma ferramenta funcional e configurável para a Trust Systems, acrescentando uma técnica de anonimização às já existentes e aplicadas pela empresa.

6.2 Dificuldades e Limitações

No que diz respeito aos desafios, o primeiro ponto a destacar é o facto de o tema deste projeto, Privacidade Diferencial, ser, até então, totalmente desconhecido por mim e por toda a equipa da Trust Systems. Sendo um conceito matemático bastante complexo, a correta implementação de todo o sistema exigiu um esforço significativo de investigação e compreensão, de forma a garantir que a transposição da parte teórica para a prática fosse realizada da melhor forma possível.

Além disso, surgiu a necessidade de utilizar múltiplas tecnologias, linguagens e ferramentas novas, entre as quais a linguagem Go (até então nunca utilizada por mim), bem como o Traefik e toda a sua relevância no contexto da empresa, além do Redis e Flyway. Nunca tinha tido contacto com estas ferramentas, o que implicou um período de aprendizagem e a necessidade de recorrer ao apoio de outros membros da empresa, em vez de conseguir iniciar o trabalho de forma imediata.

Passando para as limitações, um ponto importante a referir é a dependência do utilizador para a configuração de parâmetros críticos, um processo ao mesmo tempo importante e complexo. A solução requer que o utilizador configure manualmente certos valores, como os limites dos dados e o mecanismo a selecionar, o que pode conduzir a resultados incertos caso os parâmetros não sejam corretamente definidos, introduzindo um potencial ponto de falha na implementação.

Outro aspeto relevante a destacar é que, uma vez que a solução se baseia na interceção de pedidos, o sistema não tem acesso direto à base de dados. Consequentemente, certos dados que poderiam ser obtidos automaticamente através dessa fonte, acabam por ter de ser fornecidos manualmente pelo utilizador, o que implica uma confiança excessiva nas informações introduzidas.

6.3 Trabalho Futuro

Com base em todo o trabalho desenvolvido, existem algumas direções a seguir para o trabalho futuro, que permitiriam expandir e melhorar as capacidades da solução desenvolvida neste projeto:

- **Desenvolvimento de uma Interface Gráfica (*Frontend*):** A criação de uma interface intuitiva e de fácil utilização tornaria a ferramenta muito mais acessível. Além de melhorar a experiência de uso, essa interface poderia incluir assistentes para a configuração de parâmetros, explicações sobre o significado de cada opção, visualização dos dados antes e após a anonimização e, ainda, um possível painel de controlo para a gestão de orçamentos.
- **Expansão de Mecanismos e Estatísticas:** A integração de novos mecanismos de Privacidade Diferencial, acompanhada do suporte a funções estatísticas adicionais (como medianas ou desvios padrão), aumentaria a utilidade da ferramenta em diferentes contextos de análise.
- **Seleção Automática de Mecanismos:** Apesar de ser uma abordagem bastante complexa, a implementação de um módulo capaz de sugerir ou aplicar automaticamente o mecanismo mais adequado, com base no tipo de dado e no contexto de utilização, reduziria a carga de configuração por parte do utilizador e aumentaria a eficiência de toda solução desenvolvida.

Bibliografia

[1] Krishna Acharya, Franziska Boenisch, Rakshit Naidu, and Juba Ziani. Personalized differential privacy for ridge regression, 2024.

[2] Muhammad Aitsam. Differential privacy made easy, 2022.

[3] Samrawit Asseffa and Bihil Seleshi. A case study on differential privacy. In *Unknown Conference*, 2017.

[4] Atlassian. Sourcetree - git gui, 2024. `https://www.sourcetreeapp.com/`.

[5] Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising, 2018.

[6] Farouk Ben. Api response time standards: What's good, bad, and unacceptable, 2024. `https://odown.com/blog/api-response-time-standards/?utm_source=chatgpt.com`.

[7] M.A.P. Chamikara, P. Bertok, Ibrahim Khalil, D. Liu, and Seyit Camtepe. Local differential privacy for deep learning, 08 2019.

[8] Tore Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift*, 15:429–444, 1977.

[9] Roxana Danger. Differential privacy: What is all the noise about?, 2022.

[10] DataCamp. O que é anonimização de dados? técnicas, ferramentas e práticas recomendadas explicadas, 2025. `https://www.datacamp.com/pt/blog/what-is-data-anonymization`.

[11] Inc. Docker. Get started with docker, 2024. `https://docs.docker.com/get-started/docker-overview/`.

[12] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 1–12, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[13] Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. Differential privacy in practice: Expose your epsilons! *Journal of Privacy and Confidentiality*, 9(2), Oct. 2019.

[14] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers, 2014.

[15] Ulfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS'14, page 1054–1067. ACM, November 2014.

[16] Shaistha Fathima, Rohith Pudari, and Jacob Merryman. Local vs global differential privacy, 2021.

[17] Mario Filho. Mape em machine learning, 2023. `https://mariofilho.com/mape-erro-absoluto-percentual-medio-em-machine-learning/`.

[18] Raina Gandhi. Technology factsheet: Differential privacy. Belfer Center for Science and International Affairs, Harvard Kennedy School, 2020. Fall 2020.

[19] Gonzalo Munilla Garrido, Xiaoyuan Liu, Florian Matthes, and Dawn Song. Lessons learned: Surveying the practicality of differential privacy in the industry, 2022.

[20] Google. Differential privacy library, 2024. `https://github.com/google/differential-privacy`.

[21] DSAID GovTech. Protecting your data privacy with differential privacy: An introduction. *Medium*, 2020.

[22] Miguel Guevara. How we're helping developers with differential privacy, 2021. `https://developers.googleblog.com/en/how-were-helping-developers-with-differential-privacy/`.

[23] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential privacy under fire. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 33, USA, 2011. USENIX Association.

[24] Yuval Harness. What is differential privacy?, 2022. `https://dualitytech.com/blog/what-is-differential-privacy/`.

[25] Michael Hilton and Cal. Differential privacy : A historical survey. In *Unknown Conference*, 2012.

[26] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. Diffprivlib: The ibm differential privacy library, 2019.

[27] IBM. Ibm differential privacy library, 2024. `https://github.com/IBM/differential-privacy-library`.

[28] Apple Inc. Differential privacy overview, 2024.

[29] Traefik Labs. Traefik. https://traefik.io/.

[30] Tumult Labs. Tumult Analytics, December 2022. https://tmlt.dev.

[31] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115, 2007.

[32] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. L-diversity: privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 24–24, 2006.

[33] Microsoft. Introduction to plug and play. https://learn.microsoft.com/pt-br/windows-hardware/drivers/kernel/introduction-to-plug-and-play.

[34] Microsoft Corporation. Differential privacy for everyone, 2012. https://download.microsoft.com/download/D/1/F/D1F0DFF5-8BA9-4BDF-8924-7816932F6825/Differential_Privacy_for_Everyone.pdf.

[35] Mudit Gaur. Employee dataset, 2025. https://www.kaggle.com/datasets/gmudit/employer-data/data.

[36] Joseph P. Near and Chiké Abuah. *Programming Differential Privacy*. Self-published, 2024.

[37] Hiep Nguyen, Jong Kim, and Yoonho Kim. Differential privacy in practice. *Journal of Computing Science and Engineering*, 7, 09 2013.

[38] OpenDP. Opendp library, 2024. https://github.com/opendp/opendp.

[39] OpenMined. Pipelinedp. https://github.com/OpenMined/PipelineDP.

[40] VPN Overview. Big data e privacidade, 2024. https://vpnoverview.com/pt/privacidade/navegacao-anonima/big-data-privacidade/.

[41] Pipedrive. O que é crm? guia de gestão de clientes, 2025. https://www.pipedrive.com/pt/blog/o-que-e-crm.

[42] Inc. Postman. What is postman?, 2024. https://www.postman.com/product/what-is-postman/.

[43] PyTorch. Opacus: Differential privacy library for pytorch. https://github.com/pytorch/opacus, 2024.

[44] Google Research. Tensorflow privacy, 2024. https://github.com/tensorflow/privacy.

[45] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. In *Desconhecido*, 1998.

[46] Anshu Singh. Practitioner's guide to accessing emerging differential privacy tools. `https://medium.com/dsaid-govtech/practitioners-guide-to-accessing-emerging-differential-privac\protect\penalty\z@y-tools-861acddb2a44`.

[47] Trust Systems. Trust systems, 2024. `https://www.trustsystems.eu/`.

[48] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David O'Brien, Thomas Steinke, and Salil P. Vadhan. Differential privacy: A primer for a non-technical audience. *ChemRN: Computational Materials Science (Topic)*, 2018.

[49] Shiliang Zhang, Anton Hagermalm, Sanjin Slavnic, and Elad Michael Schiller. Evaluation of open-source tools for differential privacy. *Sensors*, 23(14), 2023.

Apêndice A

Ficheiro `apply.go`

```go
package applydp

import (
    "bytes"
    "compress/gzip"
    "context"
    "crypto/tls"
    "encoding/json"
    "fmt"
    "io"
    "log"
    "net/http"
)

func decompressIfGzip(resp *http.Response) ([]byte, error) {
    var reader io.ReadCloser
    var err error

    if resp.Header.Get("Content-Encoding") == "gzip" {
        reader, err = gzip.NewReader(resp.Body)
        if err != nil {
            return nil, fmt.Errorf("failed to create gzip reader: %v", err)
        }
        defer reader.Close()
    } else {
        reader = resp.Body
    }

    return io.ReadAll(reader)
}

type Config struct{}

func CreateConfig() *Config { return &Config{} }

type ApplyDP struct {
    next   http.Handler
    name   string
    config *Config
}

func New(ctx context.Context, next http.Handler, config *Config, name string) (
    http.Handler, error) {
    return &ApplyDP{
```

```
44          next:   next,
45          name:   name,
46          config: config,
47      }, nil
48  }
49
50  func (a *ApplyDP) ServeHTTP(rw http.ResponseWriter, req *http.Request) {
51      client := &http.Client{
52          Transport: &http.Transport{
53              TLSClientConfig: &tls.Config{
54                  InsecureSkipVerify: true,
55              },
56          },
57      }
58
59      // Reconstruct full URL from intercepted request
60      targetScheme := "https"
61      targetHost := req.Host
62      targetPath := req.URL.RequestURI()
63      targetURL := fmt.Sprintf("%s://%s%s", targetScheme, targetHost, targetPath)
64
65      // Copy the request body (it can only be read once)
66      var bodyBuf bytes.Buffer
67      if req.Body != nil {
68          _, err := io.Copy(&bodyBuf, req.Body)
69          if err != nil {
70              http.Error(rw, "Failed to read request body", http.
                    StatusInternalServerError)
71              return
72          }
73      }
74      req.Body = io.NopCloser(&bodyBuf)
75      bodyCopy := io.NopCloser(bytes.NewReader(bodyBuf.Bytes()))
76
77      // Create new request with copied method, headers, and body
78      newReq, err := http.NewRequest(req.Method, targetURL, bodyCopy)
79      if err != nil {
80          http.Error(rw, "Failed to create new request", http.
                StatusInternalServerError)
81          return
82      }
83      for key, values := range req.Header {
84          for _, value := range values {
85              newReq.Header.Add(key, value)
86          }
87      }
88
89      newReq.Header.Set("Accept-Encoding", "gzip")
90
91      // Send request
92      respOriginal, err := client.Do(newReq)
93      if err != nil {
94          http.Error(rw, fmt.Sprintf("Error forwarding request: %v", err), http.
                StatusBadGateway)
95          return
96      }
97      defer respOriginal.Body.Close()
98
99      // Decompress if needed
100     body, err := decompressIfGzip(respOriginal)
101     if err != nil {
```

```
102         http.Error(rw, fmt.Sprintf("Failed to read or decompress response: %v",
                err), http.StatusInternalServerError)
103         return
104     }
105
106     // Attempt to decode the JSON
107     var data interface{}
108     if err := json.Unmarshal(body, &data); err != nil {
109         http.Error(rw, fmt.Sprintf("Invalid JSON from target: %v", err), http.
                StatusInternalServerError)
110         return
111     }
112
113     log.Printf("Original response data (before applying privacy): %v", data)
114
115     // Send to DP service
116     payload := map[string]interface{}{
117         "endpoint": req.URL.Path,
118         "method":   req.Method,
119         "data":     data,
120     }
121     jsonPayload, err := json.Marshal(payload)
122     if err != nil {
123         http.Error(rw, "Failed to marshal payload", http.
                StatusInternalServerError)
124         return
125     }
126
127     respDP, err := http.Post("http://api:8000/apply_dp", "application/json",
            bytes.NewReader(jsonPayload))
128     if err != nil || respDP.StatusCode != http.StatusOK {
129         http.Error(rw, "Error calling apply_dp service", http.
                StatusInternalServerError)
130         return
131     }
132     defer respDP.Body.Close()
133
134     dpBody, err := io.ReadAll(respDP.Body)
135     if err != nil {
136         http.Error(rw, "Failed to read apply_dp response", http.
                StatusInternalServerError)
137         return
138     }
139
140     rw.Header().Set("Content-Type", "application/json")
141     rw.Write(dpBody)
142 }
143
144 func (p *ApplyDP) PluginName() string { return "applydp" }
145
146 func (p *ApplyDP) DisplayName() string { return "Apply Differential Privacy" }
147
148 func (p *ApplyDP) Summary() string { return "Differential Privacy" }
149
150 func (p *ApplyDP) TestData() map[string]string {
151     return map[string]string{
152         "test": "data",
153     }
154 }
```