PM

Avaliação de Implementações da Tecnologia de WebSockets

PROJETO DE MESTRADO

André Igor Freitas Figueira
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

dezembro | 2020

Avaliação de Implementações da Tecnologia de WebSockets

PROJETO DE MESTRADO

André Igor Freitas Figueira

MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO

Leonel Domingos Telo Nóbrega

Agradecimentos

Este projeto de mestrado é o resultado de muitas horas de trabalho e não podia deixar de exprimir os meus sinceros agradecimentos a algumas pessoas que me ajudaram a trilhar este percurso, pois em alguns momentos sem eles não seria possível alcançar esta meta e atingir, deste modo, mais um patamar da minha vida.

Começo por agradecer, especialmente aos meus queridos pais que sempre me transmitiram amor, bons conselhos, ensinaram a lutar pelos meus objetivos e por suportarem a minha instrução.

Aos meus avós por todo o amor, educação, e todos os momentos que a vida me proporcionou junto deles, estarei eternamente grato por tudo o que fizeram por mim.

À minha companheira e futura mulher Mariana Malho por todo o amor, confiança e apoio.

Aos meus tios, primos, sogra e restante família por toda confiança e suporte emocional que foi imprescindível à conclusão desta etapa.

Aos meus tios Maria Pais e Rego Mendes por toda hospitalidade, carinho e conhecimento passado durante toda a estadia em Lisboa.

Aos meus amigos pelo apoio e por acreditarem sempre em mim.

Ao meu orientador, o Professor Doutor Leonel Domingos Telo Nóbrega, pelo apoio, pelas sugestões e por guiar-me ao longo do desenvolvimento deste projeto.

Por fim, quero agradecer à empresa TEKEVER pela oportunidade e a todos os colegas que me integraram e com quem tive a oportunidade de contactar no decorrer desta experiência.

Resumo

A comunicação em tempo real, constitui a troca quase concomitante de informação, sobre qualquer tipo de serviço de telecomunicação desde o emissor até ao recetor, numa ligação com uma latência insignificante. Comunicações deste tipo podem ser *half-duplex*, ou *full-duplex*.

A utilização de *WebSockets* prende-se com a necessidade de resolver os problemas do tráfego e da latência que as soluções tradicionais de comunicação em tempo real apresentam. Esta tecnologia intenciona que se obtenham recursos de forma automática: assim que o servidor os recebe, propaga-os para os clientes, sem que estes efetuem novos pedidos de recursos. A sua utilização resulta num baixo consumo de recursos da rede. Além disso, este protocolo possui comunicações bidirecionais, o que permite que o servidor e o cliente comuniquem em simultâneo, sem interrupções.

Este projeto realizou-se em contexto de estágio e teve como finalidade verificar, de entre três bibliotecas de servidor de *WebSockets*, qual apresentava melhor desempenho, principalmente perante cenários cuja carga de dados era mais elevada. Para que se levasse a cabo o projeto da melhor forma, procedeu-se a uma testagem de três bibliotecas em quatro cenários cuja carga de dados era distinta, recorrendo a duas ferramentas.

A solução implementada ao longo do presente estudo utilizou o protocolo de comunicação de *WebSockets*, devido ao facto de ser uma tecnologia realmente poderosa e profícua para o desenvolvimento de soluções baseadas na comunicação em tempo real.

Palavras-Chave

WebSocket; Mensagens Instantâneas; Sistemas em Tempo Real; Bibliotecas de Software; Teste de Software; Desempenho de Software.

Abstract

Real Time Communication (RTC) is considered as the almost simultaneous exchange of information about any type of telecommunication service from the sender to the receiver, in a connection with negligible latency. Communications of this type can be half-duplex or full-duplex.

The use of WebSockets is related to the necessity of solving network traffic and latency problems, which are presented by the traditional solutions of Real Time Communication. This technology has the intention of obtaining resources automatically: as soon as the server receives them, it sends them to the clients, without the customer placing new requests. The utilization of WebSockets results in a low consumption of network resources. Besides, this protocol has bidirectional communications, that allow the communication between server and customer, simultaneously, without interruptions.

This project took place in the context of an internship and it has the goal of verify, between three WebSockets server libraries, which one presented better performance, mainly towards sceneries with higher load. In order to perform this in the best way, the three libraries testing was carried out in four sceneries whose data load was distinct, using two tools.

The implemented solution during the study used the WebSockets communication protocol, due to the fact that this technology is powerful and useful for the development of solutions based on Real Time Communication.

Keywords

WebSocket; Instant Messaging; Real-Time Systems; Software Libraries; Software Testing; Software Performance.

Índice de conteúdo

Índice de Figuras

Índice de Tabelas

Lista de abreviaturas e acrónimos

AJAX – *Asynchronous JavaScript and XML*

API – *Application Programming Interface*

C# – C Sharp

CMS – Content Management System

CSS – Cascading Style Sheets

DB – Database

ER – Entity-Relationship

FE – Front-end

GUI – Graphical User Interface

GUID – Globally Unique Identifier

HTML – *Hypertext Markup Language*

HTTP – *Hypertext Transfer Protocol*

IDE – Integrated Development Environment

IRC – *Internet Relay Chat*

JS – JavaScript

KPI – Key Performance Indicator

POC – Proof of Concept

QA – Quality Assurance

RTC – *Real-time Communication*

SSE – *Server-Sent-Events*

SSL – *Secure Sockets Layer*

TCP – *Transmission Control Protocol*

TLS – *Transport Layer Security*

URL – *Uniform Resource Locator*

VU – Virtual User

W3C – *World Wide Web Consortium*

WebRTC - *Web Real-Time Communication*

WS – *WebSockets Protocol*

WSS – *WebSockets Secure Protocol*

1. Introdução

Atualmente, a *internet* é utilizada de forma muito recorrente pelo ser humano para o consumo de informação e partilha da mesma. Ainda há pouco tempo, eram utilizados os *blogs* e os fóruns para a partilha de opiniões, dicas, tutoriais, fotografia, e muitos outros conteúdos. Com o tempo e o surgimento de novas tecnologias e plataformas como é o exemplo das aplicações móveis, passou-se a utilizar as redes sociais, e com elas as diferentes funcionalidades de partilha ou consumo de conteúdo. Estas funcionalidades assentam, por exemplo nos comentários, nas reações (*like* / *dislike* e entre outros), nos *stories*, nos vídeos em direto, nos *marketplaces* e ainda nos grupos ou salas de conversação. Todas as funcionalidades que as aplicações deste género apresentam, têm como principal objetivo manter o utilizador ativo na aplicação durante a maior fração de tempo possível. Esta permanência de atividade leva ao interesse, por parte do utilizador, de obter os conteúdos mais atualizados, e no menor intervalo de tempo possível.

O ser humano dos dias de hoje, vive com a necessidade de estar constantemente atualizado. Esta vontade não acontece apenas nas redes sociais, mas também em outras áreas como a IoT para as *smart homes* e ainda para os jogos multijogador.

O surgimento da tecnologia *WebSockets* veio tornar mais próximo do real aquilo que se considerava a atualização de dados em tempo-real, pelas vantagens que esta tecnologia apresenta em relação às utilizadas outrora. Esta tecnologia compreende diversas bibliotecas, no entanto nem todas possuem as mesmas caraterísticas, principalmente em termos de desempenho.

Neste documento estudam-se três de bibliotecas de servidor de *WebSockets,* cuja linguagem de programação utilizada para a sua implementação assentou na C *Sharp* (C#). O estudo das mesmas prendeu-se com a sua performance, dado que se realizaram uma panóplia de testes para cada uma das bibliotecas recorrendo à utilização de duas ferramentas, de modo que existisse mais que um resultado para cada cenário testado.

1.1. Contexto de estágio

Este projeto propõe-se a apresentar o trabalho efetuado em estágio. O mesmo foi realizado na empresa TEKEVER, entre setembro de 2018 e junho de 2019.

À data do início do estágio, a TEKEVER apresentava interesse em desenvolver um servidor baseado em *WebSockets*, capaz de fornecer notificações e atualizações para as aplicações desenvolvidas para as suas empresas clientes. Devido ao desconhecimento desta tecnologia, efetuou-se uma pesquisa com o objetivo de avaliar e entender o seu funcionamento e, consequentemente, aplicou-se um pequeno exercício visando um melhor entendimento acerca do tema.

1.1.1. Apresentação da TEKEVER

A TEKEVER foi fundada em 2001 e desde então a sua atividade debruçou-se na especialização de produção de tecnologias, produtos e serviços em diversas áreas, nomeadamente, nas Tecnologias de Informação e Comunicação, Aeronáutica, Espaço, Defesa e Segurança. Em 2006, esta empresa começou a internacionalizar-se. Além disso, tem ganho liderança em vários consórcios de investigação, como a vigilância do mediterrâneo

para a Agência Europeia de Segurança (AES), e o *Space Wireless sensor networks for Planetary Exploration* (SWIPE) para a Agência Espacial Europeia (AEE).

1.1.2. Contexto de integração do estágio na TEKEVER

No presente estágio foi imprescindível a integração numa equipa responsável pelo desenvolvimento de um projeto na vertente de desenvolvimento de *software*, chamado de Transformação Digital dos Produtos e Serviços, para uma empresa cliente.

A empresa em questão é uma agência sem fins lucrativos, vocacionada para o desenvolvimento de um ambiente de negócios competitivo, que procura promover a globalização da economia portuguesa. As suas principais atribuições têm como objetivos promover a internacionalização das empresas portuguesas, apoiando a sua atividade exportadora, captar o investimento estruturante e promover a imagem de Portugal com iniciativas criadoras de valor para o nosso país.

1.2. Motivações

Uma das principais motivações que proporcionou a realização deste projeto acerca deste tema, neste estágio, deve-se ao potencial da tecnologia *WebSockets*, que levou à curiosidade em adquirir conhecimento acerca deste protocolo de comunicação e da sua utilização em tempo real. Além disso, experienciar o contacto com o mundo profissional revelou-se uma grande oportunidade, dado que o mesmo seria enriquecedor e potenciador de conhecimento prático que, por sua vez, se aliaria aos conhecimentos teóricos adquiridos anteriormente.

Outra das razões relacionou-se com a necessidade, por parte da TEKEVER, de desenvolver um novo módulo que possibilitasse a notificação de novas atualizações ou correções aos canais aplicacionais dos produtos dos seus clientes. Além do que, aquando da realização do estágio curricular, de entre os projetos administrados pela empresa, o que incluía a implementação de um servidor baseado em *WebSockets* era um dos mais interessantes.

1.3. Questão de pesquisa

Nesta secção, dar-se-á ênfase à questão de pesquisa que orientou o projeto, do ponto de vista da empresa e da área da engenharia.

Até à data não se verifica que a tecnologia *WebSockets* tenha tido uma utilização significativa, o que é incompreensível tendo em conta todas as vantagens que esta apresenta, principalmente para sistemas que requerem atualizações de informação contínua e com atrasos ínfimos. Desta forma, foram estudadas três bibliotecas de servidor baseado em *WebSockets* numa tentativa de compreender qual a mais adequada através dos resultados provenientes dos testes de desempenho.

Assim, o projeto efetuado teve como objetivo responder à seguinte questão de pesquisa:

Das bibliotecas de servidor baseado em WebSockets estudadas, em qual se verifica uma melhor efetivação dos testes de desempenho?

1.3.1. Relevância da questão de pesquisa para a empresa

No projeto foi definido o desenvolvimento de uma aplicação *web*. O acesso à mesma podia ser efetuado através de duas vertentes: a de utilizador e a de gestor. Nesta, um cliente podia realizar um pré-registo, que implicaria que um gestor tomasse uma decisão de aprovação ou não do pedido de acesso. No caso de ter sido aceite, este tornar-se-ia utilizador e poderia consultar informação vantajosa e útil para a(s) empresa(s) a que o mesmo se encontrasse associado. No que concerne à vertente de gestor, este poderá gerir o conteúdo que um utilizador consulta, através de uma outra plataforma.

Este projeto surgiu da necessidade de obter uma perceção apurada, uniforme e atualizada da oferta da agência, que assentava na oportunidade de as empresas poderem realizar negócios internacionais. A comunicação encontrava-se fragmentada e inconsistente – fosse por imprecisão na categorização dos produtos e serviços, por redundância do seu conteúdo relativamente a outros ou por mera desatualização – com a dupla finalidade de aprimorar a organização interna da agência e permitir a perceção clara do impacto nesta oferta que decorre da referida transformação digital.

Desta maneira, no ato da apresentação dos requisitos exigidos à TEKEVER, um deles assentava no fornecimento de notificações aos utilizadores, em casos específicos como o alerta de uma nova mensagem, ou de um evento que decorresse em breve, configurando uma situação adequada à utilização da tecnologia de *WebSockets*.

1.3.2. Relevância do ponto de vista da área da engenharia

Do ponto de vista da área da engenharia este tema apresentou-se como sendo relevante, uma vez que, subjacente ao desenvolvimento do projeto em torno do tema, existia uma identificação em termos de tecnologias de comunicação, especialmente para a *web*. Esta identificação era referente às bibliotecas de *WebSockets* atuais, as quais apresentavam um suporte contínuo e com maior número de *downloads.* Desta forma, concluir-se-ia, através dos resultados provenientes dos testes de desempenho, que biblioteca apresentava melhores resultados. Este projeto teve um foco especial para bibliotecas desenvolvidas sob a linguagem de programação C#, devido ao facto de esta ser a linguagem de servidor utilizada pela empresa.

Ademais, o presente projeto permitiu que a apresentação de um estudo de três bibliotecas, fosse centralizado num só documento. É importante referir ainda que aquando da realização dos testes de desempenho, todas as bibliotecas experienciaram diferentes condições de utilização, desde ambientes cujo fluxo de dados e o número de ligações era elevado ou a situação contrária.

1.4. Objetivos

O principal objetivo estabelecido para esta avaliação foi o desenvolvimento de uma solução que permitisse a execução de testes de desempenho, através de ferramentas de testes, sob cada uma das bibliotecas de *WebSockets,* em diferentes condições, de modo a compreender qual delas demonstrava melhores resultados em termos de desempenho, de forma a selecionar uma biblioteca a ser utilizada para satisfazer a necessidade de integração de um servidor baseado em *WebSockets* nos projetos das empresas clientes da TEKEVER. A biblioteca selecionada, representará uma mais-valia para a empresa, pois será capaz de fornecer notificações e atualizações para as aplicações desenvolvidas, já que a empresa tenciona reduzir o tempo de entrega de informação aos canais aplicacionais.

Assim, passar-se-á à definição dos objetivos para este projeto:

1. Desenvolver uma solução que permita executar testes de desempenho;
2. Desenvolver uma aplicação de chat para o canal *web;*
3. Identificar a biblioteca com melhores resultados;
4. Utilizar a biblioteca escolhida para desenvolver um servidor de *WebSockets*.

1.5. Metodologia

Numa fase posterior à definição dos objetivos, e dada a complexidade do projeto, foi necessário estabelecer uma metodologia de desenvolvimento para delinear a ordem de trabalhos que iriam decorrer.

A avaliação das bibliotecas selecionadas ocorreu à *posteriori* do término do desenvolvimento de uma solução que integrasse as bibliotecas, bem como seleção das ferramentas de testes.

Desta forma, o desenvolvimento de uma solução que proporcionasse esta integração, foi realizado de forma simples e incremental, de maneira que se tivesse perceção do que já estava implementado e integrado. Deste modo, dividiu-se o projeto em diferentes etapas de implementação: primeiramente, a criação e estruturação de uma base de dados (*Database* – DB) relacional para o armazenamento da informação, utilizando uma abordagem *Code First* da *Entity Framework (EF);* posto isto, a implementação das bibliotecas de servidor de *WebSockets*; de seguida, a criação dos métodos nas classes comportamentais responsáveis pela lógica de negócio; em quarto lugar, o aprimoramento do design aplicacional; e por fim, o refinamento da solução colocando em prática a utilização dos padrões e princípios de desenho.

Para a implementação da solução acima descrita, manuseou-se o *Integrated Development Environment* (IDE) *Microsoft Visual Studio 2019 Community*, utilizando a linguagem de programação C#. Utilizou-se uma programação orientada a objetos, fazendo uso de um tipo específico de projeto: o *Console Application* para a implementação das bibliotecas de servidor baseado em *WebSockets.* Para fins de conetividade e realizações de operações de criação, leitura, atualização e remoção (*create, read, update and delete* – CRUD) sobre a DB, foi utilizada a EF.

Em seguida, foram identificados os critérios que auxiliaram a seleção das ferramentas de automação de testes que foram utilizadas para avaliar as bibliotecas de *WebSockets*, tendo em consideração o objetivo do trabalho e o suporte WS das ferramentas encontradas.

Os testes realizados tiveram como finalidade a obtenção de relatórios com gráficos e estatísticas das métricas de desempenho recolhidas durante o processo de teste. Em primeiro lugar definiram-se os casos de teste, também nomenclados por cenários, esta definição consistiu na identificação das características e condições de cada teste. A etapa seguinte, esteve relacionada com criação dos *scripts* responsáveis pela simulação da execução dos cenários anteriormente estabelecidos. Posto isto, foi feita uma recolha de informação dos

relatórios correspondentes a cada cenário, e também construídas duas tabelas com os dados resultantes das três bibliotecas testadas, para cada uma das ferramentas.

1.6. Estrutura do projeto

Este estudo está dividido em 8 capítulos, que se estruturaram da seguinte forma: introdução, contexto tecnológico, análise, desenvolvimento da solução, resultados, conclusões, referências e anexos.

No primeiro capítulo: a introdução, contextualizou-se o projeto definindo as motivações que levaram à escolha do tema, identificou-se a questão de pesquisa que está na origem do desenvolvimento do projeto, definiram-se os objetivos subjacentes à mesma e estabeleceu-se a estrutura do documento. Para o segundo capítulo, referente ao contexto tecnológico iniciou-se com uma contextualização histórica da comunicação em tempo real. Neste capítulo também se estudaram algumas das principais técnicas de comunicação *web*. Numa etapa seguinte, descreveu-se o protocolo HTTP, bem como uma linha temporal acerca dos mecanismos utilizados para inovar a comunicação *web*. Já nas subsecções foram descritas as tecnologias utilizadas para o desenvolvimento da solução, bem como os tipos de testes e ferramentas que foram utilizadas para este projeto.

No capítulo alusivo à análise, refere-se quais os requisitos do sistema, qual o desenho arquitetural utilizado, bem como quais os diagramas que se utilizam no processo de análise.

No quarto capítulo, em que se aborda o desenvolvimento da solução, faz-se uma descrição sumária do processo de desenvolvimento, referindo todas as decisões que foram tomadas, e descrevendo toda a experiência do desenvolvimento e de testagem de uma solução.

Posteriormente, aborda-se os resultados. Neste capítulo demonstra-se os testes que se efetuaram à solução desenvolvida, tendo sido estes os de desempenho.

No próximo capítulo, realizam-se algumas considerações finais em jeito de conclusão. Neste capítulo constam todas as conclusões que resultaram da realização do projeto, através de um processo reflexivo relativamente a todo o desenvolvimento e tendo em conta os resultados obtidos. Ademais, aborda-se a aplicabilidade da solução e descrevem-se possíveis trabalhos futuros que poderão ser realizados tendo em vista melhores resultados.

No capítulo seguinte constam as referências utilizadas na elaboração deste projeto.

Por fim temos o capítulo dos anexos, onde constam algumas figuras ilustrativas da aplicação de chat para o canal *web*. As figuras anexadas permitem comprovar as funcionalidades que esta aplicação constitui.

2. Contexto Tecnológico

Neste capítulo far-se-á uma apresentação do estado da arte e analisar-se-á, com recurso à revisão da literatura, o tema selecionado: os *WebSockets*. Com isto pretende-se identificar e clarificar quais as opções existentes e atuais, de modo a satisfazer a necessidade de desenvolvimento de uma solução para a questão de pesquisa apresentada em 1.3.

Numa primeira fase, elaborar-se-á uma contextualização histórica, recorrendo a uma linha temporal dos protocolos de comunicação *web*, sejam estes de transmissão de dados em tempo real ou não. Posto isto, pretende-se explicitar algumas técnicas e mecanismos utilizados pelos protocolos de comunicação, sendo que muitas destas continuam a ser utilizadas, atualmente. Assim, numa fase posterior, passar-se-á ao estudo dos *WebSockets* que foi, de facto, o foco deste projeto. De seguida, apresentar-se-ão os testes de *software*, onde se abordarão as necessidades da realização dos mesmos, o ciclo de vida de um teste de *software*, a abordagem do mesmo e as técnicas utilizadas para cada tipo de teste. Por fim, abordar-se-ão as ferramentas de testes de *software*.

Segundo Pandey e Bein, a Comunicação em Tempo Real (*Real-time Communication – RTC*) assenta numa parte importante da convergência das redes modernas. Os conteúdos de vídeo, voz ou texto são frequentemente utilizados em serviços de rede IP e em aplicações, sendo que muitos serviços *web* utilizam RTC. Todavia, para consumir estes conteúdos da forma mais atualizada possível, é necessário que se realizem *downloads* e que se utilizem aplicações nativas, *plug-ins* ou extensões. Enquanto a comunicação em tempo real não acontecer de forma mais simples, não haverá margem para evolução ou inovação no meio de interação *web* [1].

Para Zhangling e Mao, a *web* tornou-se disponível para a sociedade atual, quer através dos computadores, quer dos *smartphones*. Esta tem se revelado a forma mais recorrente no que respeita à acessibilidade dos conteúdos na *internet*, tanto no ramo comercial, como no pessoal. A título de exemplo, o desenvolvimento de um serviço de comunicação em grupo é considerado uma boa maneira de melhorar a acessibilidade ao serviço *web*, sendo que poderá ser executado de maneira a que não se necessitem de características específicas, ao contrário da maior parte dos serviços *web* que recorrem aos RTC, como foi referido anteriormente [2].

Tendo em conta os autores supracitados, a comunicação em grupo trata-se de uma troca de informação entre os grupos de participantes numa sessão. O desenvolvimento de serviços de comunicação em grupo na *web* é um excelente modo de melhorar a acessibilidade da RTC. Este tipo de comunicação pode ser dividido em duas categorias: assíncrona e síncrona. A comunicação assíncrona é facilmente implementada, através da utilização de formulários *Hypertext Markup Language* (HTML) e sistemas de *back-end* (BE), como é o caso da integração de bases de dados. No que concerne à comunicação síncrona, esta é considerada como o exemplo mais fiel de uma RTC, sendo que a comunicação entre o emissor e o recetor acontece de forma contínua, independentemente dos mesmos estarem no mesmo lugar [2].

2.1. Dados históricos

As aplicações *web* e as nativas que necessitam de ligação à *internet,* requerem um meio para comunicar e transferir dados entre os anfitriões (emissor e recetor). Existem três tecnologias de comunicação em tempo real padronizadas pelo HTML5 *Web Apps Working*

Group, sendo estas: os *WebSockets* [3], os *Server-Sent Events* (SSE) e os *Web Real-Time Communication* (*WebRTC)* [4].

Com base em Estep, a tecnologia *WebRTC* não alcançou a sua maturidade, nem uma extensa utilização. Assim, das três tecnologias de comunicação enunciadas, procedeu-se a uma filtragem mantendo-se o estudo acerca dos *WebSockets e os SSE.* Estas permitem a entrega de dados em tempo real, quer seja numa comunicação unidirecional ou bidirecional, com uma sobrecarga e estrutura mínimas. Estas tecnologias foram desenhadas para obtenção de uma eficiência máxima com grande escalabilidade e com o objetivo de serem compatíveis com as infraestruturas da rede, sendo que são também, conciliáveis com os *browsers* atuais mais utilizados, incluindo as versões mobile dos mesmos [4].

Para o autor supramencionado, o *web browser* tem se tornado, progressivamente, numa plataforma com capacidade de produzir inúmeras aplicações interativas. Muitas destas são do tipo *groupware* ou *collaborative software* e as mesmas possibilitam a colaboração entre empresas, pois permitem que diferentes equipas trabalhem em conjunto de forma eficiente e contribuem para que estejam todos atualizados, o que implica a existência de uma ligação persistente entre os colaboradores. Estas ligações persistentes, também conhecidas por *always-on* são, atualmente, muito utilizadas. As aplicações deste tipo exigem uma constante ligação à *internet* com o objetivo de que estas sejam atualizadas de forma dinâmica, garantindo assim o seu bom funcionamento. Isto não seria possível em tempos passados, já que para se visualizar nova informação ter-se-ia de recorrer a atualizações constantes da página [4].

O número de aplicações *web* e a suas capacidades têm vindo a crescer muito, desde aplicações de escritório, como *Google Docs*, das mensagens instantâneas, como *WhatsApp* ou o *Slack*, até aos jogos de multijogador, como o *Runescape*. Pode observar-se que, independentemente do tipo de aplicação, estas necessitam de ser alimentadas com maior quantidade de dados e frequência, de forma a corresponder às expectativas dos utilizadores. A demanda pela entrega do conteúdo de forma instantânea, tem sido cada vez mais solicitada, conforme a utilização crescente das aplicações que utilizam estes serviços. Segundo os autores Estep e Matt West, a principal função do paradigma da comunicação em tempo real é a entrega dos dados, da forma mais rápida possível, ao destino, assim que os novos dados se encontrem disponíveis [4], [5].

Segundo Hickson, com o aparecimento da tecnologia *WebRTC*, tornou-se possível a realização de chamadas de voz, vídeo e mesmo a partilha de ficheiros de um *browser* para outro, sem que existisse a necessidade de se instalar qualquer *plug-in* adicional. Porém, a mesma não alcançou a sua maturidade, nem uma extensa utilização, como já foi referido, assim se se comparar com as outras tecnologias referidas, verifica-se que estas são muito recorrentes para comunicações uni e bidirecionais, uma vez que são de simples implementação no *browser* e altamente escaláveis [6].

A tecnologia SSE possibilita que um *browser* receba atualizações automáticas do servidor via ligação *Hypertext Transfer Protocol* (HTTP). Esta tecnologia é padronizada como parte do HTML5 pelo *World Wide Web Consortium* (W3C), que explicita como os servidores podem iniciar a transmissão de dados para o cliente assim que uma ligação tenha sido estabelecida pelo mesmo. Esta tecnologia é comumente utilizada para enviar atualizações de mensagens ou *stream* de dados contínuos para o *browser*, além de ser empregue para

aprimorar o *streaming* nativo e *inter-browser* através de uma *Application Programming Interface* (API), em *JavaScript*, designada *EventSource* da qual, um cliente tem a possibilidade de efetuar um pedido de um *Uniform Resource Locator* (URL), em particular, de modo a receber um *event stream*. Hickson refere ainda que esta tecnologia se reduz a um limite de seis ligações concorrentes por *browser*, o que pode ser desfavorável quando se pretende utilizar vários separadores no mesmo. A grande vantagem, tal como nos *WebSockets*, assenta no facto de ser utilizada apenas uma ligação *Transmission Control Protocol* (TCP) [7], [8].

De acordo com Michael Carter, o protocolo *WebSockets (WebSockets Protocol* – WS*)* foi inicialmente referenciado na especificação HTML5 por *TCPConnection* como forma de substituir a API *socket TCP-based*. Foi em junho de 2008 que surgiu a primeira versão do protocolo tal como o conhecemos atualmente, designado por *WebSockets*. Segundo os logs de IRC, este protocolo apareceu a partir de uma série de discussões guiadas por Michael Carter numa sala do *Internet Relay Chat* (IRC). Tendo sido posteriormente integrado este padrão nas especificações do HTML5 [9], [10]. Para West, este protocolo de *WebSockets* representa um grande passo na evolução da *Internet*, à semelhança do que aconteceu por volta do ano 2000, aquando da mudança de paradigma com o surgimento da tecnologia AJAX. Tendo os *WebSockets* a capacidade de abrir uma ligação bidirecional, com baixa latência, estes aparecem, deste modo, como desbloqueadores de uma nova geração de aplicações *web* em tempo-real [5]. Segundo Fujishima e Ukai, o *Google Chrome* 4 foi o primeiro *browser* a fornecer suporte ao padrão RFC 6455 onde os *WebSockets* estão ativos por defeito [11].

2.2. Tecnologias *Web*

Para West, através do protocolo HTTP tradicional, não existe margem para evolução devido ao seu mecanismo de pedido-resposta, no qual é o cliente quem inicia uma transação. O modelo de comunicação, subjacente ao HTTP, não foi desenhado para suportar comunicações em tempo real. Assim, de forma a superar esta limitação, uma panóplia de diferentes estratégias foi desenvolvida. Com isto também se pretendia a superação do modelo *web* página por página e possibilitar que os servidores pudessem submeter dados para os clientes, sem que estes tivessem efetuado o pedido [5].

Conforme Estep, a simulação de uma comunicação *full-duplex* sobre um protocolo *semi-duplex,* normalmente, requer que sejam mantidas duas ligações: uma para o *upstream* e outra para o *downstream*. Este tipo de abordagem ainda é recorrente pela maioria dos serviços *web,* e o expectável é que não seja descontinuada num futuro próximo. Estas técnicas são conhecidas por *polling*, *long-polling*, e *streaming*, sendo que são baseadas na utilização de tecnologias como a AJAX (*Asynchronous JavaScript and XML*) e ainda da tecnologia *Comet*, que é também conhecida por *AjaxPush* [4].

Segundo Fierro, com o surgimento da AJAX por volta de 2005, iniciou-se a exploração da possibilidade de realizar ligações bidirecionais entre cliente e servidor, sem sucesso. Embora a AJAX tenha sido um avanço tecnológico e um ponto de partida para as atualizações de dados em tempo real, esta é uma tecnologia de comunicação unidirecional, já que, é necessário efetuar um pedido ao servidor para obter nova informação em vez de recebê-la diretamente através de um envio efetuado pelo servidor [12].

De acordo com Marchetto, Tonella *et al.*, a tecnologia AJAX trata-se de um conjunto de tecnologias utilizadas para simplificar a implementação de aplicações *web* dinâmicas. Através da AJAX, os programadores podem implementar uma comunicação assíncrona entre o cliente e o servidor. Para que esta seja possível, o objeto *XMLHttpRequest* é responsável pelo envolvimento de qualquer pedido de serviço entre o cliente e o servidor. A resposta assíncrona de um pedido é tratada do lado do cliente através de código *JavaScript*, que é despoletado pela resposta do servidor [13].

Segundo os autores supracitados, o protocolo de sincronismo de pedido-resposta, utilizado pelas aplicações *web* tradicionais, possui um atraso entre o pedido de uma página e a página seguinte. Este protocolo impossibilita que os tempos de resposta das interações sejam reduzidos. Já nas aplicações que utilizam a tecnologia AJAX, os pedidos são assíncronos, o que significa que a interface gráfica de utilizador (*Graphical User Interface – GUI*) permanece ativa e responsiva. Além desta vantagem, pode verificar-se a existência de uma outra que torna esta tecnologia útil para o desenvolvimento *web*, que assenta no facto de a mesma possibilitar a atualização de informação da página de forma dinâmica, uma vez que tais atualizações acontecem em componentes distintos da página e de forma independente [13]. Gutwin, Lippold *et al.* aludem que a tecnologia em questão provou ser um sucesso para o fornecimento de conteúdo dinâmico, pela sua falta de necessidade de atualização das páginas. No entanto, o envio de dados desde o *browser* para o servidor, e a requisição de novos dados por parte do *browser*, exigem que sejam criadas uma ligação e uma mensagem por cada atualização, o que constitui um gasto de recursos elevado [14].

De acordo com Nadaf *et al.* e Xuan *et al.*, a *Comet* é uma tecnologia utilizada na implementação de aplicações *web* para efetuar comunicações entre o cliente e o servidor. Esta é também conhecida por HTTP *Server Push*, *Two Way Web*, *Reverse AJAX*, HTTP *Streaming* e ainda *AJAX Push.* Esta tecnologia é caraterizada pela baixa latência da sua transmissão de dados, que decorre da técnica de permanecer aberta uma ligação HTTP de longa duração, também conhecida por HTTP *long-polling*. Porém, verificou-se que a abordagem da tecnologia *Comet* não é vantajosa, porque o facto manter uma ligação aberta até que o servidor tenha algo para enviar, poderá causar no lado do cliente, no *browser*, um bloqueio de novos pedidos até que os pedidos anteriores tenham sido devolvidos com uma resposta. Além disso, do lado do servidor, uma ligação aberta resultará num consumo elevado de recursos [15], [16]. Tal como refere West, com o tempo, foram detetadas desvantagens da tecnologia *Comet* e foi comprovado que este a tecnologia é ineficiente para mensagens pequenas devido ao facto de a mesma funcionar sobre o protocolo HTTP e à dimensão do *header* que um pedido desta natureza acarreta [5]. Além disso, de acordo com Peter *et al.*, muitas soluções de *streaming Comet* recorrem ao *long-polling*, caso um servidor *proxy* seja detetado a armazenar dados em *buffer* [17].

2.3. Mecanismos HTTP

Conforme menciona Rees, as aplicações *web* foram originalmente desenvolvidas em torno do modelo cliente-servidor, em que o cliente *web* era sempre quem iniciava as transações e os pedidos de dados ao servidor. Deste modo, não existia nenhum mecanismo que permitisse que o servidor desempenhasse o papel de iniciador da transação [18].

Loreto *et al.* bem como, Pimentel e Nickerson destacam que de forma a tentar mudar este paradigma foram desenvolvidos vários mecanismos com a finalidade de melhorar o modelo. Assim, o servidor poderia enviar dados ao cliente, sem que este tivesse efetuado um

pedido prévio. A tradicional técnica de *short-polling*, também conhecida por *polling*, consiste numa sequência de pedidos e respostas de mensagens. O cliente inicia o processo enviando um pedido ao servidor, o qual recebe o seu pedido, respondendo com uma nova mensagem se existir uma resposta, caso contrário envia uma resposta sem conteúdo, ou seja, espera-se que exista um *pull* de uma mensagem com dados ou então vazia [19], [20].

Para Loreto *et al.*, após um curto intervalo de tempo (Δ), designado de intervalo *polling*, o cliente volta a enviar um pedido para averiguar a existência de alguma nova mensagem disponível. Neste sentido, este processo de alerta de uma nova mensagem é realizado pelo cliente, que por sua vez questiona ao servidor, procurando obter informação. Este período em que o cliente fica a aguardar antes de reenviar o pedido é conhecido por frequência *polling* e essa frequência depende da latência que o cliente tolera para readquirir a informação atualizada reenviada pelo servidor [19].

Segundo Pimentel e Nickerson, várias aplicações como *chats* e jogos *online* utilizam a tecnologia HTTP *polling*, já que esta é considerada uma boa solução para a entrega de informação em tempo real, no caso de se saber o intervalo de entrega de cada mensagem, ou seja, quando a taxa de transmissão de dados é constante, como é o caso da transmissão de dados provenientes das leituras de sensores de níveis de água ou temperatura horária. No entanto, em situações em que a taxa de transmissão aumenta, existe uma repetição inerente aos *headers* dos pedidos HTTP *polling*, o que resulta num aumento significativo da latência [20].

De acordo com Gutwin *et al.*, o *long-polling* possibilita que o servidor atualize o *browser*, com o envio de uma resposta, quando tem novo conteúdo para ser enviado. Desta maneira, não há a necessidade de o *browser* estar continuamente a efetuar novos pedidos ao servidor a requisitar atualizações, o que resultará na redução do número de ligações e de mensagens HTTP [14]. Esta técnica foi desenvolvida a partir da necessidade de comunicação em tempo real, uma vez que simula uma indisponibilidade do servidor de modo a manter uma ligação HTTP aberta. Neste contexto a ligação entre o servidor e o *browser* (cliente) fica em *stand-by* até que sejam alcançados os parâmetros de espera (*time-outs*), ou até que o servidor tenha novos dados para enviar. Segundo West e Gutwin *et al.,* assim que os dados sejam recebidos ou a ligação seja encerrada devido a um *time-out*, o *browser* restabelece a ligação e aguarda pela próxima atualização, daí que esta técnica permita alcançar algo próximo de uma comunicação *full-duplex* [5], [14]. Tal como referem os autores Gutwin *et al.*, para esta técnica de *long-polling* a utilização dos recursos da rede é realizada de forma mais eficiente do que na técnica de *polling*, em circunstâncias nas quais os dados são enviados de forma não frequente pelo servidor e os dados provenientes do *browser* têm de ser entregues no menor tempo possível [14]. Pimentel refere que a utilização do *long-polling* arrecada algumas contrapartidas, como é o caso da sobrecarga do *header* HTTP, pois cada pedido ou resposta HTTP *long-polling* representa uma mensagem HTTP [20]. Todavia, segundo Gutwin *et al.*, observa-se também a existência de uma desvantagem na técnica de *long-polling*, perante situações que requeiram atualizações em tempo real, verifica-se uma equivalência do desempenho desta técnica ao *polling* [14].

Relativamente ao mecanismo HTTP *Streaming*, segundo Pimentel, este mecanismo mantém uma ligação aberta de forma não definitiva, sendo que nunca é terminado o pedido nem encerrada a ligação, mesmo depois de o servidor ter enviado os dados para o cliente. Este mecanismo reduz significativamente a latência na rede, uma vez que a ligação é sempre

a mesma entre cliente-servidor o que não implica abrir e fechar a ligação a cada pedido. O processo do estabelecimento de uma ligação para este tipo de mecanismo/ técnica HTTP é realizado através de um pedido iniciado pelo cliente, o qual fica a aguardar por uma resposta. Neste sentido, o servidor adia o envio de uma resposta, como acontece na técnica de *long-polling*, até que exista uma atualização disponível, ou, na pior das hipóteses, até que ocorra um *time-out*. Assim que existam dados a serem enviados, o servidor devolve a atualização como parte da resposta ao cliente. Porém, este envio dos dados pelo servidor ao cliente não faz com que o pedido ou a ligação terminem e o servidor volta ao estado em que coloca o pedido em espera até que exista uma atualização para ser enviada para o cliente. Assim, a ligação permanece aberta com o objetivo de entregar futuras mensagens [20]. No entanto, uma vez que este mecanismo é encapsulado sob o protocolo HTTP, com o aumento da latência da entrega dos dados, intervenientes como *firewalls* e servidores *proxies* poderão fazer *buffering* da resposta. Existem também soluções *Comet* que quando detetam que está a ser feito *buffering* alteram o mecanismo para *long-polling*, como já foi referido.

2.4. Transmissão de Dados através do Protocolo *WebSockets*

Tendo em conta Zhandling e Pimentel *et al.*, ao contrário do protocolo HTTP, o protocolo WS permite estabelecer uma ligação bidirecional persistente de um só *socket* TCP entre cliente e servidor, possibilitando, desta forma, que estes possam trocar conteúdos a qualquer momento, proporcionando, então, a atualização de conteúdos de forma dinâmica. Desta maneira, os *WebSockets* são considerados uma alternativa viável à tecnologia *Comet*, uma vez que evitam os problemas de ligação e portabilidade, que se verificavam nesta tecnologia. Além disso, os *WebSockets* providenciam melhores soluções que as AJAX *polling* [2], [20].

Os *WebSockets* despoletaram uma nova geração de aplicações *web* em tempo real, já que as comunicações através deste mecanismo podem ser utilizadas por qualquer aplicação, sendo principalmente empregues na vertente das aplicações *web* [21]. De acordo com os autores Deveria *et al.*, a grande maioria dos *web browsers* lidam atualmente com *push-notifications* (atualizações de aplicações em tempo real), devido a tecnologias como a dos *WebSockets*, uma vez que esta é padronizada e suportada pela grande maioria de *browsers*, como o *Google Chrome*, *Microsoft Edge*, *Internet Explorer*, *Safari*, *Firefox* e o *Opera*, sendo que este protocolo faz parte dos padrões HTML5 desenvolvidos pelo W3C [22], [23].

O padrão *WebSockets* providência uma API que poderá ser consumida através da linguagem *JavaScript*, permitindo assim aos programadores *web* a abertura de uma ligação para o servidor, e por sua vez enviar e receber dados, segundo a Web APIs MDN [24]. Para Yemini e Wikiwand, o protocolo de WS foi padronizado pela *Internet Engineering Task Force* (IETF) e publicado e discutido em 2011 na RFC (*Request for Comments*) 6455. Ambos os protocolos *WebSockets* e o HTTP estão no modelo OSI (*Open System Interconnection*) sobre a camada da aplicação (camada número sete), sendo que dependem da mesma camada de transporte (camada número quatro, TCP). Este é um modelo conceptual, que caracteriza e estabelece um padrão de comunicação das telecomunicações e que, por sua vez, está dividido em sete camadas de abstração [21], [25].

Tal como referem Fette e Melnikov, embora estes sejam protocolos de comunicação diferentes, o padrão RFC 6455 garante que os *WebSockets* sejam designados para trabalhar sobre o protocolo HTTP através da porta 80 ou 443 TCP, no caso de ser uma conexão

encriptada TLS (*Transport Layer Security*) ou uma conexão encriptada SSL (*Secure Sockets Layer*), versão 3.0 (SSLv3) especificada na RFC 6101, e para suportar *proxies* HTTP, tornando-o assim compatível com o protocolo HTTP [26]. Segundo a Wikiwand, de forma a alcançar esta compatibilidade é realizado um acordo entre os dois protocolos, nos quais se utilizam o campo HTTP *Upgrade* do *header* para ser realizada a troca do protocolo HTTP para o WS [21]. Todavia, segundo Freier *et al.* e Barnes *et al.,* observaram-se vulnerabilidades relativamente à segurança, no protocolo criptográfico SSL na sua versão 3.0, pelo que este se encontra obsoleto desde junho de 2015 pela IETF (*Internet Engineering Task Force*) [27], [28].

De acordo com a Wikiwand, este protocolo proporciona a interação entre um *web browser*, ou outra aplicação *web*, e um servidor *web.* A utilização deste protocolo resulta num baixo consumo de recursos, facilitando deste modo a transferência de dados em ambos os sentidos. Isto só se torna possível se se estabelecer uma forma padrão para que o servidor envie conteúdo ao cliente sem um pedido prévio realizado por este, permitindo assim a troca de mensagens enquanto a conexão estiver aberta [21].

Rees refere que as comunicações bidirecionais ativadas pelos *WebSockets* permitem que o lado do cliente e o do servidor de uma aplicação possam comunicar entre eles sem qualquer interrupção, porque depois de uma resposta ter sido recebida a conexão permanece ativa. Isto também significa que o servidor está no controlo e poderá atualizar ou comunicar com todos os clientes que têm uma conexão aberta com ele, como é possível observar na Figura 1 [18].

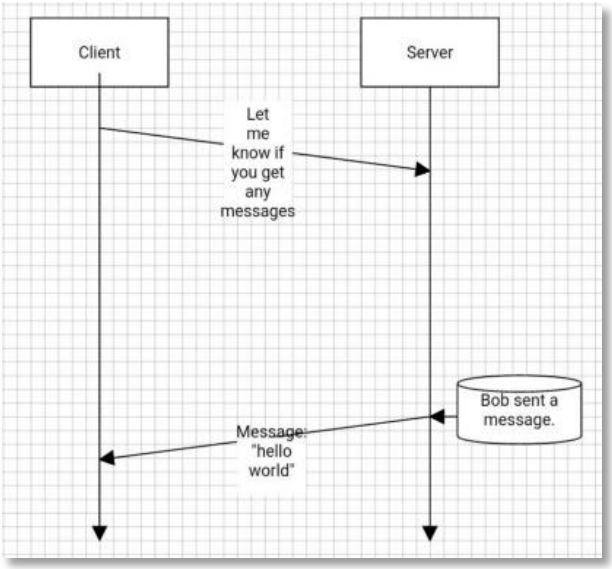


Figura 1: Representação comportamental dos *WebSockets*. [29]

Para West, uma ligação de *WebSockets* entre o cliente e o servidor é iniciada pelo cliente através de uma ação designada por *handshake*. Esta ação trata-se de um pedido HTTP do tipo GET ao servidor, com informação adicional no pedido. Esta assenta num campo *header* chamado *Upgrade* incluído no mesmo pedido, com o valor *WebSocket*, com intenção de informar o servidor acerca da pretensão por parte do cliente para permutar a ligação HTTP por uma conexão de *WebSocket*s [5].

Utilizando as ferramentas de desenvolvimento do *browser* pode observar-se o pedido de permuta com a ação de *handshake*, assim como a construção do *headers* tal como está representado na Figura 2.

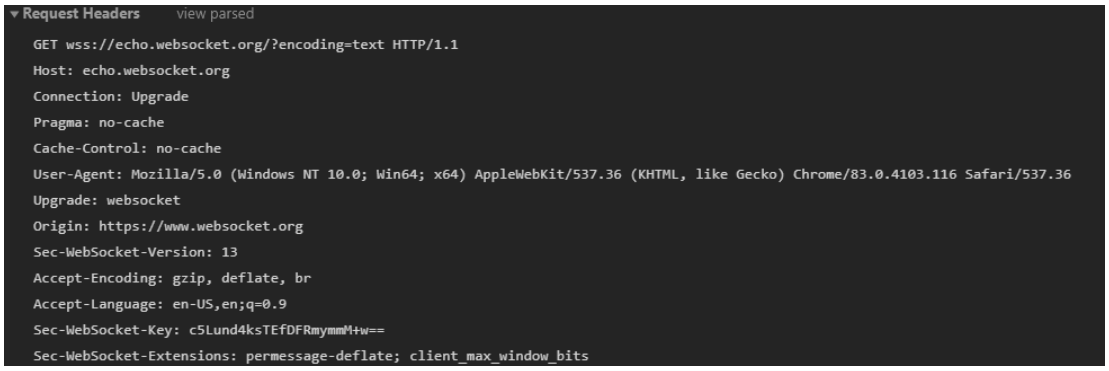


Figura 2: Pedido de upgrade para uma ligação de WebSockets (*handshake*).

A resposta de sucesso do estabelecimento da ligação de *WebSockets* pode ser observada através da Figura 3. Deste modo, Wang refere que, a ligação HTTP é quebrada, dando lugar a uma ligação *WebSockets* sobre a mesma ligação TCP/IP, sendo possível a utilização de um protocolo de encriptação ou não [30].

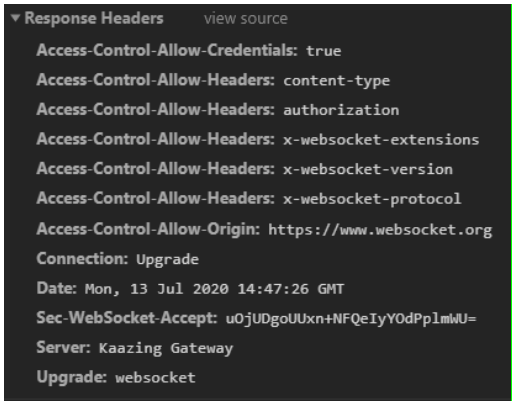


Figura 3: Estabelecimento da ligação de *WebSockets* com sucesso.

Segundo a Wikiwand, a especificação do protocolo WS define dois novos esquemas *Uniform Resource Identifier* (URI) que são aplicados para duas versões de ligações. No caso de se estar perante uma ligação desencriptada, utiliza-se o esquema WS (*que corresponde a uma ligação de WebSockets sem encriptação)* e caso a ligação seja encriptada, o esquema utilizado assenta no WSS [21].

De acordo com Rees, a tecnologia *WebSockets* pode ser utilizada em qualquer situação na qual exista uma relação cliente-servidor, porém é maioritariamente útil para aplicações que requerem constante receção e atualização de conteúdo, como por exemplo salas de conversação virtuais e atualizações constantes de informação em áreas de partidas desportivas ou financeiras. Outras áreas nas quais se observa a utilização de *WebSockets* são nos jogos multijogador *online* e na IoT (*Internet of Things*). Além das utilidades referidas anteriormente, esta tecnologia tem também uma outra aplicabilidade para as *push-notifications*, como por exemplo nos *feeds* do *Twitter* ou do *Facebook*. Aplicações do género de edição colaborativa e aplicações *online* de educação, que possibilitam a interação em tempo real entre professor e aluno, poderão, também, fazer uso deste tipo de comunicação via *WebSockets* [18].

Em síntese, os *WebSockets* vieram inovar e reduzir o tráfego de ligações que os outros protocolos de comunicação implicam, já que esta tecnologia resulta numa redução do consumo de recursos de *internet*, principalmente em dispositivos móveis. Além disso, o surgimento deste protocolo foi uma mais valia para os programadores, uma vez que este veio aumentar o número de opções da sua caixa de ferramentas, auxiliando a seleção da opção ideal, de acordo com os requisitos para o sistema a desenvolver. No entanto, como acontece em todas as tecnologias, também nesta se podem identificar um conjunto de aspetos positivos e negativos, os quais se encontram sistematizados na Tabela 1.

Vantagens	Desvantagens
Redução no tráfego na rede e na latência	Restabelecimento de ligações
Compatível com *firewalls e proxies*	Balanceamento de carga
Eficiente para atualização frequente de dados	Utilização de *frameworks*
Comunicação *full-duplex*	Escassez de bibliotecas

Tabela 1: Vantagens e desvantagens dos WebSockets.

Segundo o autor supramencionado, apesar da existência de muitas vantagens na utilização de *WebSockets*, os programadores têm de utilizar bibliotecas de abstração como as *frameworks socket.io* ou *SignalR* devido a questões relacionadas com compatibilidades ou fragmentação dos *payloads* [18]. Para Mallik *et al.*, a tecnologia de *WebSockets* é, atualmente, considerada como uma das próximas grandes tecnologias para a *web* e está a ser adotada por muitos programadores. Além do que, existem já muitas plataformas como a *Firebase* que já utilizam *WebSockets* [31].

Para desenvolver ou mesmo comprar uma solução a um fornecedor, dever-se-á ter uma visão futurista, uma vez que se deve ter em consideração a maneira de realizar as comunicações ou transportes de informação entre cliente e servidor quer a curto, quer a médio prazo, já que estas têm impacto sobre o desempenho do sistema, além do facto de que as tecnologias estão em constante evolução.

2.5. Testes de *software*

Conforme Herzog, o desenvolvimento de um produto de sucesso resume-se a alguns aspetos como: a disponibilidade, a facilidade de modificação, a funcionalidade, o desempenho e a segurança. A disponibilidade refere-se a uma propriedade do *software* que existe e que está preparada para desempenhar a sua tarefa quando necessitamos [32]. Segundo Malhotra e Chug, a facilidade de modificação está relacionada com a simplicidade com que uma alteração pode ser feita e foi observado que cerca de 60 a 70% do ciclo de vida de um produto é dedicado à sua manutenção [33]. A funcionalidade está relacionada com aquilo que aplicação permite ao utilizador realizar. De acordo com Sarojadevi, o desempenho remete para a capacidade de o sistema concluir as operações e providenciar informação de forma rápida e exata, quer para situações de elevado tráfego de informação, quer para momentos de constrangimento dos recursos de *hardware* [34]. Para McGraw, a segurança é também uma propriedade fundamental em alguns projetos. Para construir um *software* seguro, o mesmo deve ser desenhado com esse propósito. Desta maneira, deve educar-se os programadores, os arquitetos e os utilizadores para a construção de *software* que garanta segurança [35].

Conforme o que Myers refere, os testes de *software* são uma tarefa técnica, mas que também envolve algumas considerações importantes de economia e de psicologia humana, uma vez que o ser humano tende a ser determinado no alcance de objetivos, tendo em conta

que o estabelecimento dos mesmos resulta num importante efeito psicológico [36]. O ato de testar um programa assenta num processo com intencionalidade destrutiva chegando mesmo a ser considerado um processo sádico, sendo que contraria a factualidade subjacente à vida humana. Myers e Arílo destacam que a maioria dos seres humanos apresenta uma perspetiva de vida construtiva, já que, embora de forma inconsciente, se tenha a tendência de criar objetos em vez de destruí-los. Além disso, é através deste processo de tentativa de "destruição" que se visa um aumento da confiança por um produto através da exposição de problemas antes da entrega ao utilizador final [36], [37].

2.5.1. Necessidade da realização de testes

Os testes de *software* fazem parte da terceira e quarta fase do desenvolvimento do programa, e têm como principais objetivos: acrescentar valor, melhorar a qualidade e a confiança do mesmo, através da revelação das falhas no produto, fazendo com sejam identificadas as causas das mesmas e corrigidas pelas equipas de desenvolvimento. Segundo os autores Myers, Arílio e Selvapriya, deve assumir-se que os programas têm erros, e que os testes têm como intenção identificar o máximo de erros possíveis antes da entrega final, garantindo, desta forma, que os resultados estão de acordo com os padrões estabelecidos. Na terceira e quarta fases do desenvolvimento do produto, os testes são de extrema importância para a garantia da qualidade de *software*. Algumas organizações dedicam de cerca de 40% a 50%, do tempo e do orçamento, para aplicabilidade de testes [36]–[38].

De facto e conforme Myers e Jovanovic, todos os casos de um programa deveriam ser testados, no entanto, isso não é possível devido a dois constrangimentos, nomeadamente ao número infindável de casos a testar mesmo para programas simples, e por requerer muito tempo e demasiados recursos para ser economicamente viável. Portanto, a solução passa por escolher o conjunto mais adequado de casos de teste através da utilização de técnicas que estão correlacionadas com análises de risco ou, então, através da perícia das equipas de engenharia de testes [36], [39].

Na Figura 4, pode observar-se a relação entre a quantidade e a qualidade dos testes. Na mesma considera-se o número de *bugs* que não são detetados e o custo dos testes. Através desta, é percetível que quanto mais dispendiosos são os testes, menor será o número de *bugs* que não se detetam, como se pode analisar através da reta tracejada a vermelho, o contrário também pode ser observado com a reta tracejada a verde, ou seja, quanto menor é a qualidade dos testes, menos dispendiosos serão, todavia, o número de *bugs* não detetados será mais elevado. Desta forma, pode verificar-se que o ponto de interseção das duas curvas exponenciais marca a quantidade ideal de testes a efetuar, de modo a que exista um equilíbrio entre a qualidade e a quantidade dos testes. Só assim poder-se-á tomar decisões relativamente ao que se está a testar e aos testes que se devem excluir, já que se visa uma otimização do número de testes, com o intuito de reduzir o custo da testagem, ainda que não se detetem alguns *bugs*.

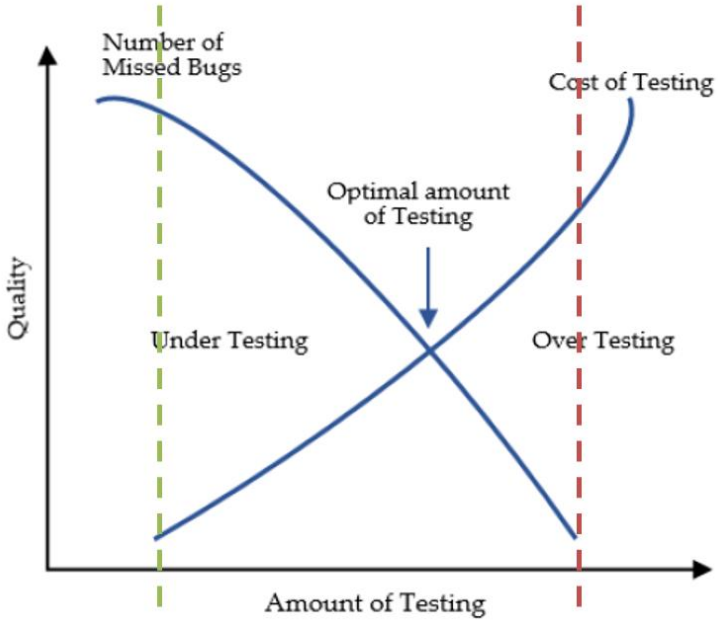


Figura 4: Relação entre a qualidade e a quantidade de testes. Elaborada por [40]

2.5.2. Ciclo de vida de um teste de software

O ciclo de testes de *software* (*Software Testing Life Cycle* – STLC), faz parte do processo de otimização dos mesmos. Segundo Parihar, é neste ciclo que são definidas todas as fases de teste de *software*. O STLC corresponde a uma das fases do ciclo de vida de desenvolvimento de *software* (*Software Development Life Cycle* – SDLC), e revela ser de uma elevada importância, uma vez que um produto final ou um *software* não deve ser lançado no mercado, sem que antes tenha passado pelo processo de STLC com sucesso [41].

No que concerne à Figura 5, a mesma apresenta as diferentes etapas do ciclo de vida dos testes percorridas por um *software* ou produto durante o processo de testes. Contudo, não existe um STLC padrão – o mesmo poderá variar em função da região na qual se encontrem os intervenientes do processo de teste.

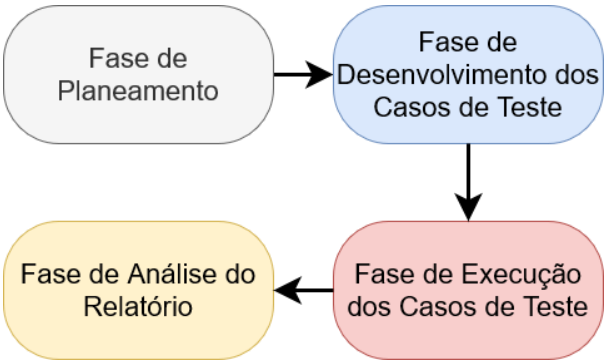


Figura 5: Ciclo de vida de testes de *software*. [40]

O ciclo de testes de *software* em questão tem como base, segundo Myers e Jamil *et al*., um plano para a execução do teste. Este plano inclui diversas dimensões como por exemplo a abrangência. Além disso, identifica a abordagem, os recursos, o cronograma das atividades de teste, bem como os itens e funcionalidades a testar, as tarefas a realizar e os riscos que estão associados à atividade do teste. Assim, pode observar-se a partir da Figura 5, que o STLC consiste em quatro fases principais, tendo o seu início na fase do planeamento e realizando um percurso até à fase na qual se analisa o relatório [36], [40].

Assim, explicitar-se-á cada uma das fases procurando detalhar cada uma delas. A primeira fase corresponde ao planeamento. Jamil *et al*. afirmam que é nesta fase que se realiza o plano de todas as atividades de teste a concretizar de forma integral [40]. A principal finalidade da fase do planeamento prende-se com a consideração de questões relativas à estratégia de teste, nomeadamente, os recursos a utilizar, as responsabilidades subjacentes aos planos elaborados, bem como os riscos e propriedades a aplicar de forma a reajustar a execução dos testes e a sua ordem. De acordo com Parihar, do fim da fase do planeamento resulta um documento que se designa de plano de teste e que permite garantir que o processo de teste é, de facto, possível [41].

Terminada a fase de planeamento, é altura de preparar o teste, o que corresponde à fase de desenvolvimento. Jamil *et al*. e Parihar referem que é neste estágio que os casos de teste são concebidos pela equipa de garantia de qualidade (*Quality Assurance – QA*) de forma manual ou, em alguns casos, de forma automatizada. Os casos de teste especificam um conjunto de valores de entrada, as condições de execução e ainda os resultados que esperados. O conjunto de dados especificado deve ser selecionado de forma a comprovar que produz os resultados de sucesso esperados, bem como, que alguns desses dados estejam intencionalmente incorretos, procurando que sejam produzidos alguns erros durante o teste, de modo a validar quais as condicionantes apresentadas pelo *software*. Esta fase engloba, ainda, outras atividades tais como: a definição dos objetivos do teste, a identificação dos elementos que serão avaliados, a preparação do ambiente no qual decorrerá o teste, e também a preparação das ferramentas que o irão suportar [40], [41].

Posto isto, passa-se, então, à terceira fase, que diz respeito à execução do caso de teste criado na fase anterior, baseando-se no plano de teste, tal como refere Jamil *et al*. [40]. Nesta fase, todos os casos de teste são executados, incluindo os de verificação e validação, lembrando que estes se referem à fase de teste, correspondente à fase de desenvolvimento. É importante salientar que os casos de teste de verificação começam no fim de cada fase do SDLC e os casos de teste de validação se iniciam depois da conclusão de um módulo. Por fim, os resultados dos testes são documentados como relatórios de incidências de teste, de *logs* e de estado e ainda como relatórios sumários. De facto, e conforme Parihar, a finalização desta etapa resulta num relatório que inclui os resultados dos testes dos casos de teste estabelecidos na fase anterior. Neste relatório, as funcionalidades que passem pela fase de execução sem falhas serão consideradas aprovadas e, portanto, casos de sucesso, e os casos de teste falhados serão entregues e considerados defeituosos [41].

A quarta e última fase é relativa à análise do relatório. Nesta fase, os programadores que levaram a cabo os casos de teste analisam os mesmos com o objetivo de encontrar defeitos. A análise pode, também, ser elaborada pelos programadores em conjunto com o(s) cliente(s), já que é imprescindível que se percecione da melhor forma o que se deve ignorar e o que deve ser corrigido, melhorado ou, simplesmente, alterado.

Os relatórios de teste são constituídos pelos relatos de falhas, e por isso, devem conter os resultados que se obtiveram a partir da fase anterior. As falhas encontradas são, desta forma, reportadas à equipa de desenvolvimento com a finalidade de que esta proceda à sua correção, assim como se pode confirmar através de Jamil [40]. De acordo com Parihar, é importante referir que é imprescindível a análise relativa a estas falhas através do *feedback* dos *testers* e do cliente. Após a correção das falhas, os programadores entregam as correções às equipas de testes. Estas correções serão integradas num novo STLC [41].

Como já foi referido, o STLC é parte integrante do SDLC, todavia aquele está encarregue das fases de teste. O ciclo de testes tem o seu início assim que os requisitos sejam concebidos. Este ciclo providencia um processo gradual que assegura a qualidade do *software*, caso as falhas sejam devidamente identificadas e corrigidas.

De acordo com o autor supramencionado, durante a fase de planeamento do teste, na qual se pode verificar o desenvolvimento do código, os *testers* e os programadores, poderão analisar e definir, concomitantemente: o âmbito do teste, os critérios de valores de entrada/ saída e os casos de teste. Isto tem como finalidade melhorar a qualidade do *software* e por conseguinte reduzir a duração do STLC, o que constitui uma mais valia para a equipa de testes, uma vez que quando o desenvolvimento terminar a equipa despenderá muito menos tempo a desenhar os casos, fazendo com que se possa iniciar, então, a execução dos testes antecipadamente [41].

2.5.3. Abordagens de teste

A realização de testes requer que se definam valores de entrada, que se execute o programa tendo como base os valores definidos anteriormente e requer, ainda, que se analisem os valores de saída. Através da Figura 6 pode observar-se o fluxo de informação que um ciclo de testes envolve. Se se comparar a configuração do *software* com a do teste, perceciona-se que a primeira requer a especificação de requisitos, de desenho e de código, enquanto que a segunda exige a definição de casos de teste, o planeamento e os procedimentos de um teste e ainda a identificação das ferramentas que se utilizam aquando da execução dos mesmos. Segundo Luo, no momento em que se avaliam os resultados dos testes, deve proceder-se a uma análise dos mesmos, comparando-os com os resultados que se expectavam [42]. Para Myers e Arílio, a análise destes resultados permite que sejam extrapolados dados estatísticos tais como a taxa de erros e os erros resultantes da execução dos testes. Esta análise permite, numa fase posterior, que se estabeleçam níveis de confiança do sistema ou, no caso de se encontrarem erros, que os mesmos sejam corrigidos, seguindo-se um novo ciclo de testes sobre essas correções [36], [37].

No que concerne à configuração do teste, os casos de teste são preparados através da utilização de diferentes técnicas com a finalidade de que se elaborem testes com uma maior eficácia. Selvapriya afirma que com o recurso a estas técnicas, a integridade do programa é preservada e são selecionadas as condições de teste que garantam uma maior probabilidade de encontrar erros [38]. De facto, e conforme Hooda e Rajender, os *testers* não conseguem prever de forma sistemática, quais os casos de teste que devem selecionar, bem como, quais as técnicas a utilizar para desenhar as condições dos testes [43]. Para Selvapriya, caso uma das técnicas combine todos os tipos de técnicas que se referiu anteriormente, poderá obter-se melhores resultados, do que se se utilizar apenas uma [38].

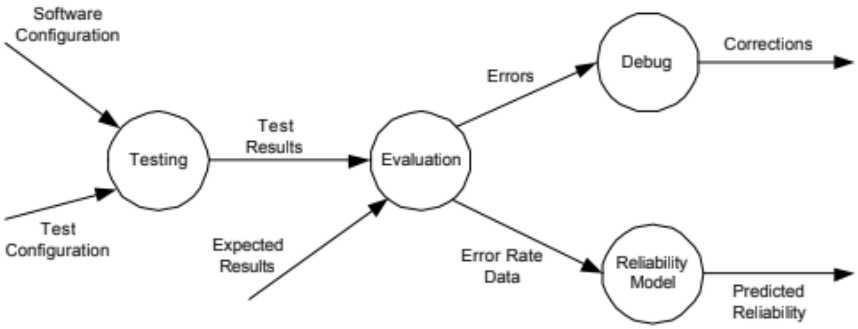


Figura 6: Fluxo de informação de um teste de *software*. [42]

2.5.4. Técnicas de teste de *software*

As técnicas de teste de *software* têm como objetivo encontrar falhas no mesmo. De acordo com Arílio, estas técnicas são selecionadas de acordo com os requisitos de teste estabelecidos. As mesmas contemplam diferentes perspetivas do *software* e devem possuir uma estratégia de teste que lhes permita tirar o máximo partido da mesma [37]. Segundo Sawant *et al.*, os testes subdividem-se em testes estáticos/ manuais e em testes dinâmicos/ automáticos, a Figura 7 carateriza os dois diferentes tipos de testes [44].

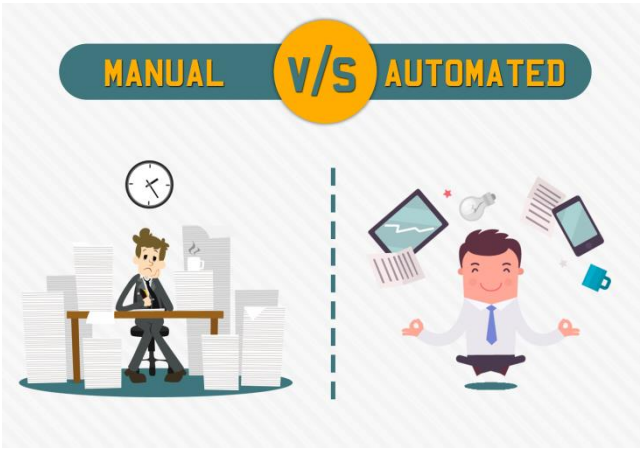


Figura 7: Testes de *software* manuais e automáticos. [45]

Os testes estáticos, também denominados como testes manuais, assentam num processo lento e trabalhoso. Este tipo de testes é aplicado, ainda numa fase inicial do ciclo de vida do *software*, pelos próprios programadores e pelas equipas de testes. De acordo com Sawant *et al.*, os testes possuem técnicas tais como: a técnica de revisão informal e a de revisão técnica [44].

Segundo o autor supramencionado, os testes dinâmicos tratam-se da automação da prática de teste. Assim, para cada teste é preparado um *script*, que é, por sua vez, executado utilizando ferramentas de testes apropriadas ao tipo de teste em questão de modo a averiguar se os requisitos de teste foram ou não cumpridos. Estes tipos de testes dinâmicos também conhecidos por testes automáticos podem ser classificados em quatro tipos de testes: os testes de correção, os testes de desempenho, os testes de confiança/ fiabilidade e os testes de segurança [44].

De maneira a sintetizar a informação referente às principais diferenças entre os dois tipos de testes, procedeu-se à elaboração da Tabela 2.

Testes Manuais	Testes Automatizados
Consomem demasiado tempo e é muito cansativo, uma vez que os casos de testes são feitos um por um, de forma manual.	São executados de maneira mais rápida que os manuais.
Envolvem um grande investimento de recursos humanos, porque para a execução deste tipo de testes é requerido muitos *testers*.	Requerem um menor investimento em recursos humanos, já que -são executados utilizando ferramentas de automatização.
São menos fiáveis, por não serem realizados com a mesma precisão devido ao erro humano.	São mais fiáveis, por serem desempenhados sempre com a mesma precisão todas as vezes que é executado o mesmo caso de teste.
São ideais para situações de execução de casos de testes para um número reduzido de vezes.	São ideais para situações de execução dos casos de teste inúmeras vezes.
Permitem que o *tester* execute mais casos de testes aleatórios.	Possibilitam que o software seja testado com diferentes configurações (testes de compatibilidade).
Apresentam um custo reduzido para curto prazo.	Apresentam reduções de custos a longo prazo.
Quanto mais tempo se dedicar a testar um módulo, maiores serão as chances de encontrar problemas de utilizadores reais.	São limitados à deteção, de defeitos expectáveis.
São mais económicos para investimentos iniciais.	São mais dispendiosos para investimentos iniciais do que os testes manuais.

Tabela 2: Comparação entre testes manuais e testes automatizados.

Cada técnica de teste tem uma aplicabilidade, sendo que cada uma é indicada para revelar diferentes aspetos de um sistema de *software*. Não havendo consenso na revisão da literatura, acredita-se que existam três grandes categorias de técnicas de teste: a técnica estrutural, a técnica funcional e a técnica não-funcional. Além disso, existe uma outra técnica (teste de caixa-cinzenta) que surgiu recentemente e que se constitui a partir das metodologias e dos benefícios das técnicas funcional e estrutural.

2.5.4.1. Técnica Estrutural

Para Jovanovic, a técnica estrutural, também conhecida por teste de caixa branca, é altamente eficaz na deteção e resolução de problemas, o que faz com que, frequentemente, sejam identificadas falhas muito antes que estas comecem a causar problemas. Esta técnica tem ainda outros nomes, é conhecida por: análise de caixa branca, teste de caixa transparente, análise de caixa branca ou, ainda, por testes baseados no código [39]. Segundo Roohullah *et al.*, a utilização desta técnica é muito importante e necessária para que o *tester* tenha uma visão e conhecimento completo de todo o código [46].

Jovanovic refere que o teste de caixa branca assenta no processo de fornecer um valor de entrada para o sistema de *software* e, ainda, no ato verificar de que maneira é que o sistema processa essa entrada para gerar o valor de saída esperado [39]. De facto, e conforme Roohullah *et al.,* o foco principal desta técnica de teste prende-se com a investigação e compreensão da lógica e da estrutura do código, como se pode observar a partir da Figura 8 [46]. Sawant *et al.* destacam que esta técnica pode ser aplicada ao nível

dos testes de *software* entre os quais se podem tomar como exemplo: os testes de integração, os unitários e os de sistema [44]. De acordo com Jovanovic, o teste de caixa branca é considerado um método de confirmação que poderá ser utilizado para a validação do seguimento para a implementação do desenho, bem como para validação das funcionalidades de segurança implementadas e, por fim, para a descoberta de vulnerabilidades [39].

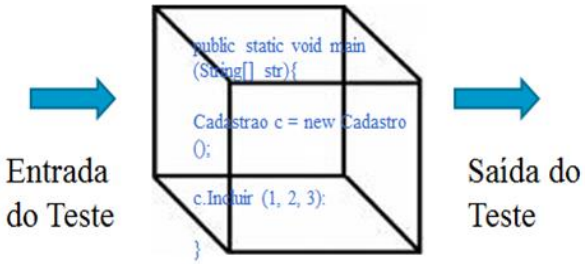


Figura 8: Teste de caixa-branca, com acesso ao código. [47]

2.5.4.2. Técnica Funcional

Segundo Sawant *et al.*, a técnica funcional, também conhecida por técnica de teste de caixa-preta, pode ser comparada a um dispositivo, cujo funcionamento é desconhecido e inacessível, uma vez que se foca apenas no comportamento externo do *software* [44]. Os *testers* ao efetuarem este tipo de testes desconhecem e são impossibilitados de aceder ao código fonte. De acordo com Roohullah *et al.*, esta técnica tem como base os valores de saída expectáveis estabelecidos nos requisitos de forma a medir o sucesso dos testes, uma vez que toda a lógica e processamento entre a introdução dos valores de entrada e os valores de saída é impenetrável e desconhecida, sendo semelhante a uma caixa negra (o que justifica uma das suas denominações) [46].

Myers refere que esta técnica é aplicável a todos os níveis de teste, no entanto não é aconselhável tentar detetar todos os erros do sistema de *software* através desta, já que a única forma de o fazer será ao introduzir todos os valores de entrada possíveis [36]. Jamil *et al.* destacam que existem três tipos fundamentais de testes de *software* para a técnica funcional, estando estes categorizados por níveis, pela ordem que se segue: os testes unitários, os testes de integração e os testes de sistema. Cada um deste tipo de testes é realizado pelos próprios programadores ou pelo *tester* de *software* [40]. Além destes três tipos existem ainda os testes de aceitação e os testes de regressão, sendo que os de aceitação são normalmente realizados pelo cliente, ou por um grupo restrito de utilizadores, enquanto os de regressão são realizados, exclusivamente, pelos *testers*.

O modelo representado na Figura 9, denomina-se modelo em V, e trata-se de um modelo SDLC, no qual são executados processos de forma sequencial, sob a forma de um V. Este modelo é também conhecido por modelo de Verificação e Validação, uma vez que a cada fase de implementação corresponde uma das etapas de teste, tendo em conta que existe apenas uma passagem para a fase seguinte e que a mesma só acontece quando a fase anterior está completa. Foi proposta a adoção de muitos outros modelos de processos de testes numa escala industrial. Arílio refere que de entre todos estes modelos, o *V-model* é o mais popular, ainda que todas as variantes se distingam pelos níveis de testes unitários, testes de integração e testes de sistema [37].

Os testes de *software*, anteriormente explanados, estão incluídos numa das etapas do SDLC. Este é um processo que garante a produção de *software* de baixo custo e simultaneamente com alta qualidade, sendo desenvolvido num curto espaço de tempo. Este processo envolve várias etapas começando pelo planeamento, passando pela construção, bem como pela realização de testes, e findando no *deployment*. Os testes devem ser definidos e aplicados de forma iterativa e paralelamente ao desenvolvimento do *software*. Além disso, estes deverão ocorrer para os diferentes níveis de teste.

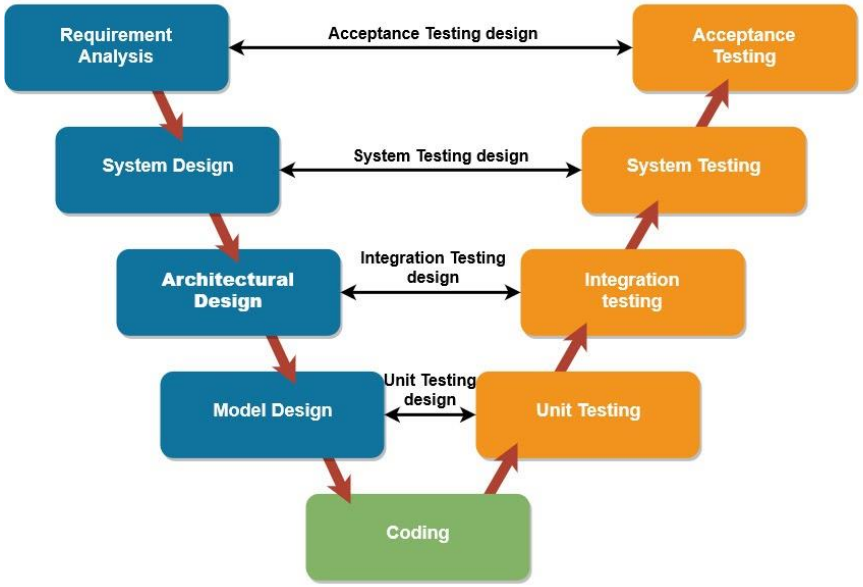


Figura 9: Paralelismo entre o desenvolvimento e os testes de *software* (Modelo em V). [37]

Segundo Carson, o planeamento dos testes deve acontecer segundo uma sequência descendente. Inicialmente planeiam-se os testes de aceitação, partindo de uma especificação de requisitos do sistema. Em segundo lugar, planeiam-se os testes de sistema, tendo em conta as especificações funcionais. Posteriormente, são planeados os testes de integração com base no *design* de alto nível do *software* (conceção geral no *software*). E em quarto e último lugar, planeiam-se os testes unitários partindo de um *design* de baixo nível (módulo específico). Enquanto que, no que concerne à execução dos testes, esta é realizada no sentido inverso. No *V-model* existem duas fases: a fase de definição, localizada no lado esquerdo e a fase de validação/ verificação, localizada no lado direito. É na fase de definição que são estabelecidos os requisitos do sistema, o seu *design*, o desenho arquitetural e o *design* do modelo. A fase de verificação e validação corresponde à fase em que são elaborados os testes, tendo como base as especificações dos requisitos estabelecidos na fase de definição, além disso, é ainda nesta fase que se efetua a avaliação do *software*, numa tentativa de perceber se o mesmo vai ou não ao encontro das expetativas do cliente. Estas duas fases encontram-se interligadas por uma outra fase: a fase de desenvolvimento de *software*, que corresponde, na Figura 9 ao *coding* [48].

Segundo a prática convencional em ambientes reais de desenvolvimento de produtos de *software*, os principais níveis de teste de *software* são, como já foi referido, os seguintes: os testes unitários, os testes de integração, os testes de sistema e os de aceitação.

Relativamente ao primeiro nível, têm-se os testes de unitários. Estes são aplicados a um baixo nível, isto é, são aplicados a pequenos componentes de uma aplicação como

métodos dos objetos, estes são realizados pelos próprios programadores, já que os mesmos escrevem o código, e executam os testes de forma a detetarem se o que foi desenvolvido se comporta como era expetável, como referem Roohullah *et al.* [46].

De acordo Sawant *et al.*, no que concerne aos testes de integração, correspondentes ao segundo nível de teste de *software*, estes assentam numa técnica que visa desvendar erros relacionados com interfaces e em consonância associar módulos previamente testados através de testes unitários transformando-os numa estrutura maior e mais próxima da estrutura do programa. Além disso, estes testes têm como objetivo testar a ligação entre os módulos [44].

Para os autores supracitados, os testes de sistema, correspondentes ao terceiro nível, não têm qualquer implicação no que se refere ao conhecimento do código, sendo que são normalmente baseados nos requisitos do sistema. Isto é, na verdade, uma sucessão de diferentes testes, que é conduzida sob um sistema completo já com a integração de todos os requisitos especificados [44]. Segundo Luo e Roohullah *et al.*, embora cada teste tenha um propósito distinto, todos trabalham com os mesmos objetivos: avaliar a conformidade entre os elementos do sistema, as especificações e a qualidade do mesmo, através da execução da aplicação como um utilizador final [42], [46].

No que respeita ao quarto e último nível, que corresponde aos testes de aceitação, também conhecidos por testes de QA, segundo os autores Luo e Sawant *et al.*, estes testes estão inseridos na metodologia de testes de caixa-preta e são avaliados a um nível geral, tendo em conta as funcionalidades do sistema e o cumprimento de todos os requisitos especificados pelo cliente. Este tipo de teste pode ser realizado de duas maneiras distintas, sendo estas: pela empresa desenvolvedora do produto ou pelo cliente/ utilizador, quando o produto é desenvolvido por uma empresa externa, sendo, por isso, considerado, pelo utilizador, um dos tipos de testes mais importantes, antes da entrega final. Salientando que atributos de qualidade não funcionais, como a confiabilidade, segurança e manutenção são, também, aferidos [42], [44].

Tendo por base Leung e White, a confirmação da confiança do programa passa por efetuar testes apenas para as novas *features* ou para os componentes modificados. Este tipo de teste pode ser aplicado em qualquer nível de teste [49]. Na Figura 10 podem observar-se os testes anteriormente descritos.

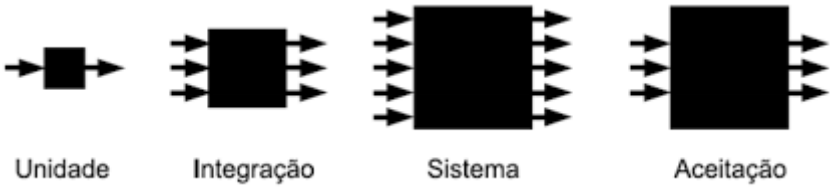


Figura 10: Teste de caixa-preta, sem acesso ao código. [50]

2.5.4.3. Técnica de Testes de Caixa-Cinzenta

Jovanovic refere que, nos últimos anos, a técnica de caixa-cinzenta tem vindo a ser considerada como mais uma técnica utilizada para testar *software* [39]. Para Roohullah *et al.*, esta técnica é, também, conhecida por caixa-translúcida, visto que combina os benefícios dos testes de caixa-branca e os de caixa-preta, além do que, a sua metodologia é independente

da plataforma e da linguagem [46]. Ademais, de acordo com Jovanovic, os testes de caixa-cinzenta são considerados imparciais e não intrusivos, pelo facto de não requererem que o *tester* tenha acesso ao código fonte [39].

Segundo Sawant *et al.,* a técnica em questão é utilizada para testar um excerto do *software* através das especificações utilizando algum conhecimento interno do seu funcionamento. Para desenhar casos de teste através da utilização desta técnica, são tidos em conta os algoritmos e as estruturas de dados internas, o que faz com que exista uma maior compreensão da lógica do funcionamento do programa, comparativamente aos testes de caixa-preta, mas uma menor compreensão se se comparar com os testes de caixa-branca [44]. De acordo com Jovanovic, verifica-se que este método é muito útil quando se pretende realizar testes de integração entre módulos de código escritos por diferentes autores, em que só é possível realizar testes através dos pontos de entrada e saída de cada módulo. Tendo em conta tudo isto, esta técnica poderá ser aplicada na grande maioria das fases de testes [39].

Leung e White mencionam que os testes de regressão se enquadram também nos testes de caixa-cinzenta, e são um tipo de teste de *software* cujo objetivo se trata de confirmar se as alterações recentes no programa não afetaram adversamente as *features* existentes. Além disso, é uma estratégia relevante pelo facto de reduzir o número de efeitos colaterais, *à posteriori* [49].

2.5.4.4. Técnica Não-Funcional

De acordo com *o site Software Testing Help*, a técnica de testes não-funcional tem como objetivo validar a qualidade do *software*. Esta técnica remete para aspetos complexos relacionados com a usabilidade, o desempenho e a segurança do mesmo. A qualidade do programa é avaliada através de métricas, de entre as quais o tempo, a precisão, a estabilidade, a integridade e a durabilidade do mesmo perante circunstâncias adversas. Além disso, a aparência, a facilidade de utilização e a segurança são também aspetos importantes quando nos referimos a esta técnica [51].

Tendo em conta a mesma fonte, em termos de *software*, quando uma aplicação funciona de acordo com as expetativas do utilizador, de forma eficiente, independentemente das condições a que é sujeito, o *software* é considerado confiável. Neste sentido, na técnica não-funcional não é praticável realizar testes manuais, já que existem ferramentas específicas de testes automáticos que são utilizadas para comprovar a qualidade do *software* [51].

Assim, tendo em conta que o presente projeto tem como finalidade última verificar o desempenho das diferentes bibliotecas de servidor baseado em *WebSockets* em circunstâncias distintas, dar-se-á lugar a uma explicitação acerca dos testes de desempenho.

Segundo Sarojadevi, os testes de desempenho são um dos tipos de testes que se inserem na técnica não-funcional, estes deverão ser realizados através da utilização de ferramentas que automatizam o processo de teste. Estas ferramentas possibilitam a configuração de casos de teste de acordo com os requisitos não funcionais do sistema, a simulação de um ambiente real de funcionamento e a produção de relatórios com os dados recolhidos dos testes [34].

Para o mesmo autor, os resultados dos testes em questão, permitem a monitorização de informações relacionadas com os níveis de utilização de recursos, tais como as percentagens de: utilização de processador, da memória, de consumo de energia e de largura de banda. Além disso, existem outros dados que podem ser recolhidos, como por exemplo: parâmetros de tempo (mais propriamente o tempo de carregamento), de resposta, de acesso, de execução, bem como, os parâmetros relacionados com a taxa de sucesso – a frequência de falhas, e o intervalo de tempo entre as falhas e o normal funcionamento [34].

Com estes dados é possível discutir se o sistema desenvolvido está em consonância com os requisitos estipulados em termos de desempenho. Além disso, permite responder a questões acerca da quantidade de utilizadores suportada pelo sistema, da forma de recuperação do mesmo quando este ultrapassa o seu limite, e ainda, o seu tempo de resposta em momentos de carga normal e de pico. Os testes de desempenho deverão ser realizados novamente a cada etapa de desenvolvimento de *software*, de modo a que se analise se as novas funcionalidades e correções tiveram algum impacto negativo no desempenho do sistema, tal como refere o autor supramencionado [34].

Hooda e Rajender referem que este tipo de testes de desempenho pode dividir-se em seis subcategorias: os testes de stress, os testes de carga, os testes *soak*, os testes de pico, os testes volume e os testes de capacidade. Destes seis tipos de testes de desempenho, os mais populares são os testes de stress e os testes de carga [43].

De acordo com Sawant *et al.*, relativamente aos testes de stress, estes são realizados para encontrar e compreender qual a capacidade máxima de um sistema como um todo, ou de apenas uma componente do mesmo. Neste tipo de testes utilizam-se ferramentas que têm como objetivo verificar se o sistema cumpre os requisitos delineados e, ainda, determinar o limite de cargas do sistema antes do mesmo apresentar uma quebra súbita [44].

No que concerne aos testes de carga, Hooda e Rajender afirmam que estes são utilizados com o intuito de determinar a robustez do sistema. Deste modo, submete-se a máquina ou o servidor a um nível de funcionamento próximo dos seus limites. Para proceder a este teste, são fornecidas ao sistema ou à aplicação inúmeras transações, visando a sua sobrecarga. Este tipo de testes poderá ser realizado em ambientes controlados de forma a que haja a possibilidade de comparar as capacidades de diferentes sistemas ou de medir individualmente, e de forma minuciosa, os seus limites [43].

Dando, neste momento, enfoque aos testes *soak* (também conhecidos por testes de resistência), os mesmos autores referem que estes são realizados com o objetivo de determinar o comportamento do sistema em termos de desempenho, através de uma carga contínua. Com a utilização destes testes poder-se-á monitorizar a utilização de memória e, por conseguinte, detetar potenciais fugas da mesma [43].

No que respeita aos testes de pico, os autores supracitados mencionam que mesmos são realizados com o intuito de observar o comportamento do sistema aquando de um aumento espontâneo do número de utilizadores, ou de uma enorme quantidade de carga gerada pelos mesmos [43].

Relativamente aos testes de volume, Hooda e Rajender aludem que estes são desempenhados visando verificar se o sistema consegue ou não lidar com uma grande quantidade de dados. Este tipo de testes são focados na DB [43].

Por fim, explicitar-se-ão os testes de capacidade, que, ainda segundo os mesmos autores, são utilizados para determinar o máximo de utilizadores que o sistema suporta sem deficiências, antes que os objetivos de desempenho comecem a ser comprometidos. Isto permite que se evitem potenciais problemas de escalonamento, seja devido ao aumento de utilizadores ou ao volume de dados [43].

Atualmente, tendo em conta Sarojadevi, perante os mercados digitais, a grande maioria das empresas possuem *websites*, e também providenciam serviços-*web*. No entanto, existem alguns problemas que estas empresas enfrentam diariamente com os seus produtos e serviços-*web*, problemas estes que resultam numa redução das receitas. Estes problemas podem ser: os tempos de carregamento excessivos das suas páginas *web* e a indisponibilidade dos seus produtos digitais. Contudo, algumas destas empresas estão a migrar os seus produtos para plataformas na *Cloud*, numa tentativa de contornar e minimizar algumas destas dificuldades. Qualquer indisponibilidade pode tornar-se deveras dispendiosa. De acordo com o relatório de Gartner, uma inatividade de uma aplicação crítica de forma não planeada, implica um custo médio em torno dos $100,000 por hora [34].

Segundo Rajni e Oak, a *Jupiter Media Metrix*, empresa líder em investigação e consultoria de mercados, que reporta o impacto da *internet* e das novas tecnologias no comércio e nas formas de comercialização, lançou um questionário que foi realizado por mais de dois mil utilizadores, e concluiu-se que perante esta amostra, cerca de 46% dos utilizadores abandonam os *websites* ou aplicações por problemas técnicos e de desempenho. Isto demonstra que os utilizadores tendem a abandonar a plataforma para este tipo de situações, em vez de providenciar *feedback* às equipas de suporte da aplicação ou *website*, fazendo com que a verdadeira causa desta falha não seja descoberta [52].

Um dos maiores constrangimentos da área tecnológica assenta no facto de algumas empresas não aplicarem testes de desempenho aos *softwares* que desenvolvem. Isto faz com que os problemas subjacentes aos mesmos não sejam detetados atempadamente e, por conseguinte, que a sua solução seja muito mais dispendiosa, tendo em conta que quanto mais tarde estes problemas são identificados, maior será o custo e o esforço para que possam ser resolvidos, tal como consigna Molyneaux [53].

Através da análise à Figura 11 que se pode encontrar num documento do mesmo autor, podem observar-se duas situações distintas. A primeira em que se observa uma linha contínua e de cor sólida, que é relativa ao que foi planeado (*Planned*) e que representa o resultado que se espera quando o processo de desenvolvimento de uma aplicação é realizado de uma forma cuidada, o que significa que o produto foi lançado com sucesso no tempo que estava estimado e cujos problemas de desempenho são ínfimos ou mesmo inexistentes. A segunda situação é representada pela linha tracejada, sendo a mesma referente ao processo mais frequente na atualidade (*Actual*), em que se dão atrasos nos lançamentos dos produtos, exigindo assim que o processo seja mais demorado, o que leva a um aumento dos custos, já que se verifica a existência de problemas de desempenho numa altura em que o produto já se encontra na fase de produção [53].

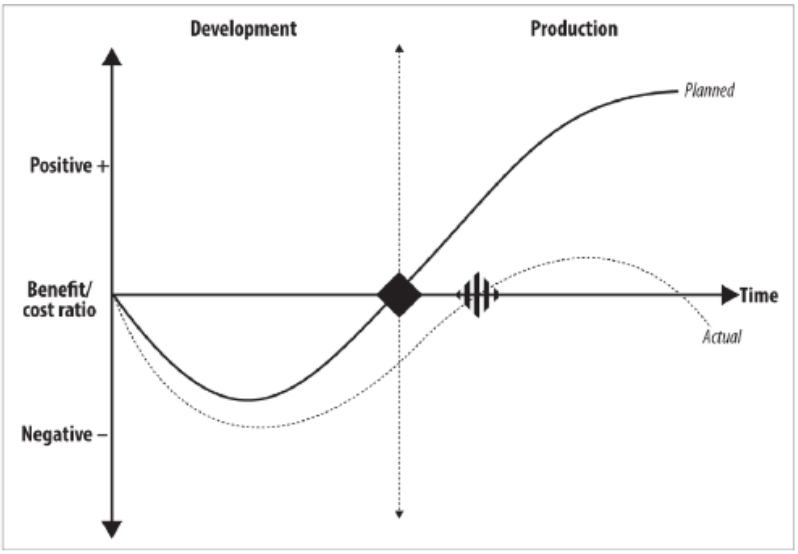


Figura 11: Curva do custo benefício do processo de desenvolvimento de *software*. [53]

Assim, pode compreender-se que, segundo Sarojadevi, tanto os *websites* como as aplicações *mobile* devem satisfazer certos critérios, tais como: o carregamento imediato das páginas *web*, o suporte de alta precisão e eficiência para as transações *online* e a garantia de uma inatividade quase nula dos seus produtos ainda antes de os mesmos serem lançados [34].

Conforme Sawant *et al.*, em todos os sistemas têm especificações de desempenho, no entanto, existem requisitos implícitos que o sistema deverá satisfazer [44]. Todavia, tendo em conta Sarojadevi, em contextos de sistemas críticos de vida, cujo seu funcionamento cause repercussões negativas na vida do ser humano, o expectável é que exista um esforço e atenção muito grande em todas as etapas da aplicabilidade deste tipo de testes. Tome-se como exemplo os sistemas utilizados ao nível dos cuidados de saúde mais precisamente em operações do foro cardiovascular e também os sistemas de descoberta de medicamentos na área farmacêutica [34].

Através da Tabela 3 pode proceder-se a uma síntese referente às diferentes técnicas de teste de *software* que se descreveram anteriormente.

Aspetos	Técnica de Teste Estrutural (caixa-branca)	Técnica de Teste Funcional (caixa-preta)	Técnica de Teste caixa-cinzenta	Técnica de Teste Não-Funcional
Aplicabilidade	Confirmar o cumprimento do *design* estabelecido, e detetar e resolver problemas	Realizar qualquer tipo de testes	Realizar testes de integração e regressão	Validar a qualidade do *software* e confirmar aspetos não funcionais do sistema
Conhecimento	Conhece-se o funcionamento interno do código	Desconhece-se o código	Conhece-se parcialmente o funcionamento interno	Desconhece-se o código/ conhecem-se os requisitos não funcionais do sistema
Execução	*Testers* e programadores	*Testers*, programadores e cliente	*Testers*, programadores e cliente	Ferramentas de testes/ *Testers*
Esforço	A mais exaustiva e a mais morosa	A menos exaustiva e menos morosa	Exaustão e morosidade intermédia	Nada exaustiva e nada morosa

Tabela 3: Comparação entre as diferentes técnicas de teste de *software*.

2.6. Ferramentas de Testes

No que concerne às ferramentas de testes, no contexto de testes de *software*, estas podem ser definidas como produtos que possibilitam a realização de uma ou mais atividades de teste. Existe todo um processo que deve ser levado a cabo para que um teste seja realizado com excelência, iniciando na fase de planeamento, passando pela fase identificação dos requisitos, posteriormente à fase da preparação do *script*, seguindo-se a execução do mesmo e por fim, a obtenção dos relatórios/ resultados onde constam os registos dos defeitos encontrados e também os gráficos que permitem uma análise mais intuitiva e visual dos resultados.

Os testes automáticos apresentam diversas vantagens se os compararmos com os testes manuais, como já foi referido. Por este motivo, tal como destaca Kaur e Gupta, diversas empresas desenvolvem ferramentas de testes automáticos para diferentes aspetos da avaliação de aplicações e diferentes tipos de teste. No entanto, estas ferramentas devem providenciar as funcionalidades de realizar testes de regressão e também testes unitários e integração [54].

A grande maioria dos provedores de serviços de testes possuem uma caixa de ferramentas que contém diversas opções, provindas de uma diversidade de fornecedores, o que lhes permite escolher a ferramenta de teste mais adequada para um requisito de teste de desempenho em particular, assim como alude Molyneaux [53]. Todavia, existem muitas subtilezas no *design web* que dificultam a tarefa de seleção de uma ferramenta, uma vez que,

estas poderão apresentar um conjunto ínfimo de funcionalidades que vão ao encontro das necessidades de *design* ou limitações que impeçam a sua escolha.

No que concerne à escolha da ferramenta de testes, de acordo com Kaur e Gupta, tendo em conta as suas capacidades e limitações, existem considerações que auxiliam a tarefa de seleção de uma forma objetiva, como é o caso da complexidade de utilização, do tempo de aprendizagem, da tecnologia, do tipo de relatórios resultantes da avaliação e do facto de esta suportar ou não o protocolo de comunicação do sistema a ser testado [54]. Além disso, Mustafa *et al.* refere que se pode observar uma consideração referente ao esforço da criação e reutilização dos *scripts*. Outra consideração sobre a qual é importante debruçar-se, prende-se com o fator económico. Este é deveras impactante e decisivo para a seleção da ferramenta de testes, o mesmo é um fator restritivo em projetos nos quais os orçamentos são limitados, dificultando, desta maneira, a escolha de apenas ferramentas gratuitas. Esta restrição do modelo de licenciamento, tem limitações que assentam no facto de o *tester* ter um número limitado de utilizadores virtuais para proceder à simulação, bem como, um número reduzido de funcionalidades, por se tratar de um licenciamento gratuito [55]. Pode também tecer-se uma consideração comparativa relativamente às ferramentas de teste, sendo que se se optar por uma ferramenta mais completa, esta será bastante abrangente e indiscutivelmente, muito dispendiosa, caso se opte por uma ferramenta cujo cariz seja mais limitador, certamente estaremos perante uma solução mais económica, porém muito menos abrangente. Existe ainda um caso especial quando se está a tratar destas ferramentas, aquando do teste de sistemas mais antigos ou sofisticados, deve usar-se uma ferramenta personalizada/ específica, desta forma a mesma acarretará mais custos.

Tendo por base Kaur e Gupta, é importante criar uma lista de requisitos subjacentes às ferramentas, além do que as mesmas deverão ser expostas a uma prova de conceito (*Proof of Concept* – POC), de modo a garantir que pelo menos uma ferramenta se encontra de acordo com os requisitos. Sem a especificação de uma lista de requisitos para a escolha da ferramenta de testes, o resultado assentará numa má gestão de tempo, sendo que é necessário que se descarreguem, instalem e avaliem a mesma, dado que esta pode atender a apenas alguns requisitos e, no pior dos casos, a nenhum [54].

Tal como já foi referido, a usabilidade da ferramenta e a criação de relatórios são, também, aspetos importantes para a comparação e escolha da ferramenta. Segundo Shahid e Ibrahim, existem ferramentas que possuem duas versões aplicacionais, uma versão com GUI e outra em modo consola (*batch*), como forma de satisfazer as diferentes necessidades dos utilizadores. A grande maioria das ferramentas de testes comerciais possuem a capacidade de produzir relatórios automáticos, de entre os quais alguns são apresentados sob a forma gráfica ou sob a forma de ficheiro (.csv) [56]. Os relatórios concebidos através de gráficos têm como base as representações estatísticas, como a média, a mediana, o desvio padrão, a distribuição normal, os percentis e os gráficos de barras, representação muito conhecida e utilizada em diversas áreas.

Para Molyneaux, a média é normalmente utilizada nos testes de desempenho para aferir o tempo médio de resposta. Apesar de existirem diversos tipos de média, para o propósito dos relatórios dos testes de desempenho a média aritmética é a representação mais apropriada. A mediana também é uma métrica interessante em situações em que a média aritmética é distorcida por um pequeno número de casos atípicos (*outliers*), o que resulta num valor que não reflete a média [53].

O desvio-padrão, representado na Figura 12, a qual é possível observar num documento do autor supracitado, é também umas das representações gráficas dos relatórios obtidos a partir dos resultados dos testes, através das ferramentas. No que respeita aos testes da técnica não-funcional, e ainda mais precisamente aos testes de desempenho, a experiência de utilizador poderá ser representada sob a forma de um gráfico de desvio-padrão. Em circunstâncias nas quais este tipo de representação indica um desvio-padrão elevado, este poderá apontar para uma experiência de utilizador imprevisível. Tomemos como exemplo, o caso de uma transação com um tempo médio de resposta de 40 segundos e cujo desvio padrão é de 30 segundos. Esta informação é suficiente para compreendermos que o utilizador final terá uma grande chance de experienciar um tempo de resposta mínimo de 25 segundos e máximo de 55 segundos para a mesma transação. Pode então afirmar-se que este tipo de representação gráfica é, deveras, percetível e sugestiva. Além do que, se deve procurar alcançar um baixo valor de desvio-padrão [53].

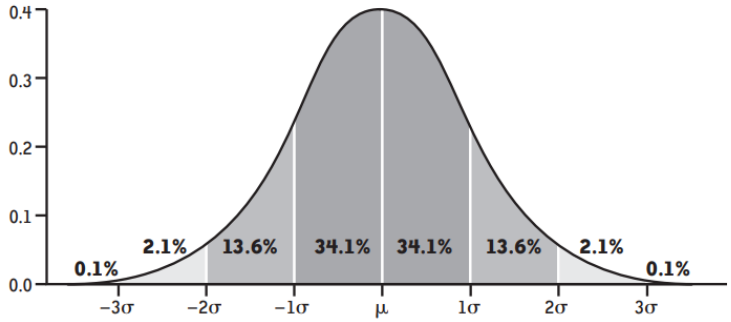


Figura 12: Modelo de distribuição normal. [53]

Os percentis são, também, importantes para os testes de desempenho, dado que estes assentam numa forma de fácil compreensão de caraterísticas de desempenho do sistema, tal como se pode observar no *site* PerfMatrix [57]. O facto de se manter um foco apropriado num percentil despoleta o fornecimento de um alicerce para automatizar a análise de desempenho da rede, de maneira a compreender o comportamento aplicacional e otimizar o sistema. O cálculo de um determinado percentil para um grupo de números não é algo linear, sendo que é necessário que se ordenem todos os tempos de resposta de forma ascendente. No entanto, as ferramentas de testes de desempenho tratam de o fazer de forma automática, sem intervenção humana. Assim, se se tiver o intuito de que o foco seja um certo percentil, deve selecionar-se o mesmo, excluindo os restantes, assim como indica Molyneaux [53]. Os valores dos percentis que se encontram nas posições 90, 95 e 99 (ver exemplo de representação na Figura 13) devem ser alvo de uma observação mais cuidada quando se está a tratar de testes de desempenho, pois estes aproximam-se mais do real.

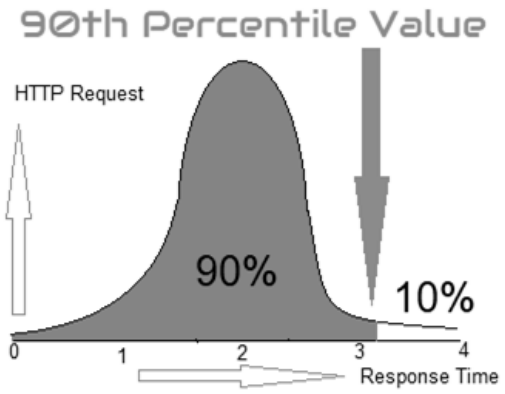


Figura 13: Representação do 90º percentil. [57]

Tendo por base o modelo de distribuição normal, os gráficos de barras são utilizados para agregar todos os tempos de resposta recolhidos durante os testes de desempenho através de intervalos de tempos de resposta. O número de barras poderá ser customizado de acordo com os ajustes dos intervalos de tempo que se pretende representar através de cada barra, normalmente sob o eixo das abcissas, já que o eixo das ordenadas indica as medidas dos tempos de resposta. A Figura 14, que se observou a partir de um documento cujo autor é Molyneaux, apresenta uma representação com recurso a um gráfico de barras criado a partir de um relatório de testes [53].

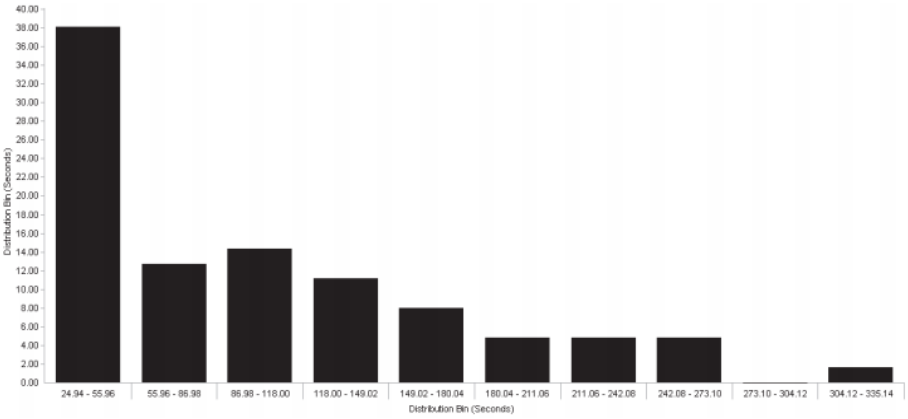


Figura 14: Gráfico de barras acerca dos tempos de resposta recolhidos. [53]

3. Análise

Neste capítulo, identificar-se-á os requisitos necessários para a concretização da solução da aplicação de chat para o canal *web*, descrita no capítulo referente ao Desenvolvimento da Solução, sendo que será efetuado um levantamento dos requisitos funcionais e não funcionais da solução e será apresentado o desenho arquitetural efetuado para a mesma.

Qualquer *software* de sucesso é construído tendo por base os requisitos recolhidos durante a fase de planeamento presente no SDLC. A definição destes requisitos tem uma grande importância já que estes servem de alicerces para o desenvolvimento do *software* e por sua vez estes irão refletir-se na qualidade do mesmo.

Este projeto visiona, do ponto de vista da empresa, a seleção de uma biblioteca de servidor baseado em *WebSockets*, para a integração da mesma na infraestrutura do produto. Não obstante, a maioria dos utilizadores interessados no sistema não estão presentes durante a fase de testes. Desta forma, a recolha de requisitos será escassa.

3.1. Requisitos funcionais

Os requisitos funcionais, segundo Summers *et al.*, focam-se na ação do sistema, definindo aquilo que este deve fazer, ou seja, define o comportamento expectável do sistema. Estes são, normalmente, definidos no início do processo e são frequentemente utilizados para conduzir a implementação do projeto [58].

Tendo por base todas as discussões e sessões de esclarecimentos com os membros da equipa integrante da empresa TEKEVER, com o orientador e toda a literatura consultada foram identificados os requisitos funcionais apresentados na Tabela 4.

Nº	Requisito funcional
RF1	Alertar todos os membros ativos quando um membro acede ou abandona a aplicação.
RF2	Transmitir a todos os membros da sala quando um membro envia algum conteúdo.
RF3	Enviar mensagens de texto e de anexos.
RF4	Criar salas de chat privadas (grupos).
RF5	Registar de novos utilizadores.
RF6	Requerer autenticação para utilizar o sistema, via *username* e *password*.
RF7	Recuperar de mensagens perdidas.

Tabela 4: Estabelecimentos de requisitos funcionais do sistema.

3.2. Requisitos não funcionais

No que concerne aos requisitos não funcionais, estes descrevem as caraterísticas do sistema ou de uma componente do mesmo, podendo incluir o tamanho, o custo, o material ou outros fatores. O principal requisito subjacente a este projeto trata-se do desempenho, tal como mencionam Summers *et al.* [58].

Na Tabela 5 podem observar-se os requisitos não funcionais identificados para a solução da aplicação de chat, sendo que estes irão definir o comportamento que o sistema terá perante a utilização dos utilizadores.

Nº	Requisito não funcional
RNF1	O sistema poderá suportar 50 mil utilizadores sem comprometer a qualidade do mesmo.
RNF2	O sistema deverá transmitir novos conteúdos até a um limite de atraso de cinco segundos.
RNF3	O tamanho limite de cada conteúdo enviado pelos membros não deverá ultrapassar os 20 MB.
RNF4	O sistema não poderá autorizar acessos sem autenticação.
RNF5	O sistema e a DB deverão estar em soluções separadas (+ segurança + facilidade para modificações)

Tabela 5: Estabelecimentos de requisitos não funcionais do sistema.

3.3. Desenho Arquitetural

Conforme Sommerville, desenho arquitetural trata-se de um processo que visa identificar os subsistemas que compõem o sistema principal e também a estrutura de controlo e comunicação dos subsistemas. Como resultante deste processo de desenho, obtém-se uma descrição da arquitetura de *software*. Este desenho arquitetural representa uma ligação entre as especificações e os processos de desenho, sendo que é muito frequente ser feito em simultâneo com algumas atividades de especificação [59].

A Figura 15 constitui o desenho arquitetural do servidor da aplicação *chat* desenvolvida para demonstrar a integração do servidor baseado em *WebSockets* com um canal. Como é possível observar-se existem três blocos principais: o bloco *Client Chat* UI, o *WebSocket Server* e a DB *Instance*. O bloco *Client Chat* UI corresponde à aplicação do canal *web* que comunica com o Chat *Server* através de uma comunicação WS.

No bloco seguinte, o *Chat Server*, é responsável pela gestão das ligações de *WebSockets*, e pela lógica do negócio, que se encontra nos módulos: *MessageRepository*, responsável pela estruturação dos dados tanto para o envio e receção dos mesmos; *HistoryData*, encarregado pela recuperação de dados perdidos; *SessionCreation*, que tem como responsabilidade a criação de novas sessões de utilizador perante uma nova ligação de *WebSockets; UserCreation,* responsável pela criação de novos utilizadores; e o *WebSocketServer* que gere todas as interações das ligações de *WebSockets*, desde a interação de pós ligação, a de uma nova mensagem enviada por um cliente, a interação de erro na ligação e a de fecho de ligação. Este bloco comunica com a instância de DB através de uma ligação MySQL. É nesta instância que se efetuam todas as operações de CRUD nas tabelas criadas segundo o Modelo de dados utilizado.

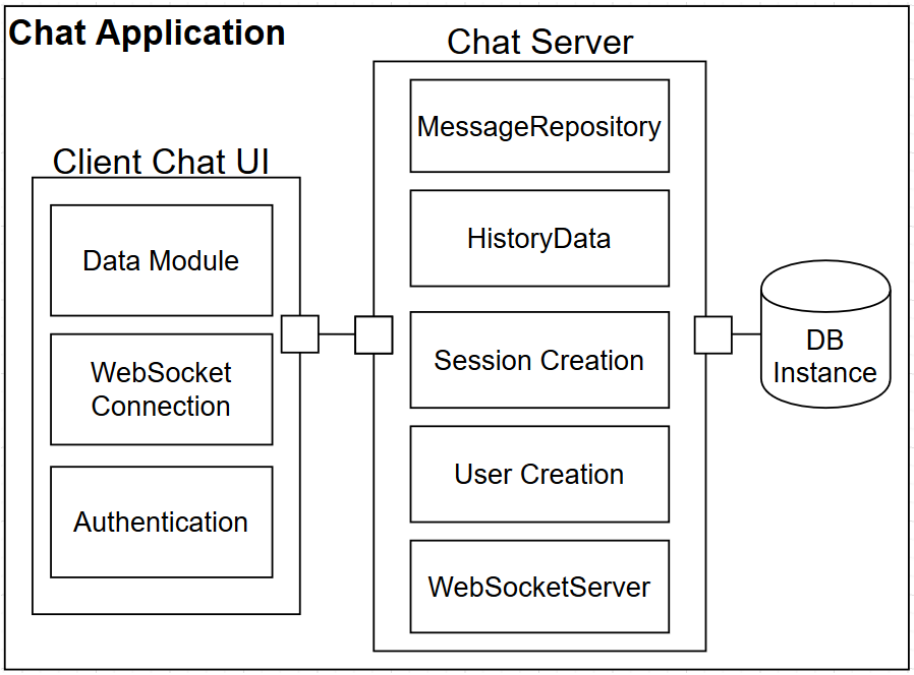


Figura 15: Desenho arquitetural da aplicação chat.

4. Desenvolvimento da Solução

Neste capítulo pretende-se demonstrar como foi realizada a implementação da solução, bem como, descrever os detalhes pertinentes para a mesma, os problemas com que se deparou ao longo do percurso, bem como as soluções encontradas para os mesmos durante todo o processo. Esta secção foi dividida em cinco subsecções nas quais se encontram:

1. A definição do modelo de dados;
2. A seleção das três bibliotecas de servidor baseado em *WebSockets*;
3. A implementação da solução;
4. O desenvolvimento da aplicação *chat* no canal *web* para demonstração do funcionamento da solução de servidor implementada;
5. A seleção das duas ferramentas de testes de desempenho e o planeamento e configuração das mesmas para a execução dos testes.

Segundo o *site* Visual Studio Magazine, para o desenvolvimento da solução a linguagem de programação utilizada foi a C# sobre a ASP.NET *Framework* 4.6.1, utilizando a tipologia de projeto *Console Application*. A escolha para esta linguagem de programação tem a ver com o facto de ser a linguagem de servidor utilizada pela empresa na qual o estágio foi realizado. Além disso, esta linguagem subiu nos *rankings* para as melhores linguagens de programação utilizadas em 2020, tal como se pode observar na Figura 16 [60].

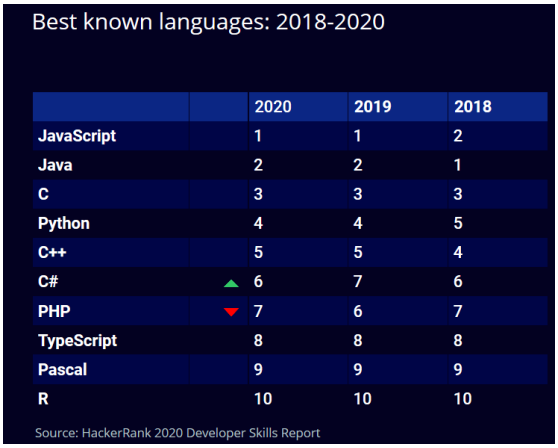


Figura 16: Ranking das melhores linguagens de programação para o ano de 2020.

Para o desenvolvimento desta solução utiliza-se a ferramenta *Integrated Development Environment* (IDE) *Visual Studio Community* 2019. A *Microsoft* possui uma ferramenta chamada *NuGet Package Manager* integrada no Visual Studio em 2010. Um *NuGet* (significa "*New Get*") trata-se de um gestor de pacotes gratuito que é responsável pelo *download*, pela instalação, pela atualização e pela configuração de *software* no sistema. Além disso é uma ferramenta *Open Source* e encontra-se envolvida num vasto ecossistema de ferramentas e serviços *Microsoft*, assim como refere Decoster [61].

4.1. Modelo de dados

Existem diversos tipos de bases de dados, no entanto para esta solução foi escolhida uma DB relacional pela sua principal caraterística: a consistência dos dados. A razão pela qual se escolheu este tipo de DB deve-se à necessidade de haver uma consistência nos dados devido ao facto de o requisito da aplicação de *chat* possuir um mecanismo de recuperação de todos os dados perdidos desde a última sessão do utilizador. Além disso, este tipo de DB tem a vantagem da facilidade de elaboração de *queries* com dados agregados. Uma *query* é uma questão, frequentemente expressa formalmente. Uma *query* à DB pode ser do tipo pedido de dados ou informação a uma tabela ou combinação de várias de DB, bem como, uma ação.

A fase de desenho do modelo de dados foi muito bem pensada e delineada, pois a estrutura (*schema*) dos dados final teve um grande impacto na consistência e na velocidade em que os dados foram gravados e obtidos da DB.

A primeira fase do planeamento consiste na identificação das entidades relacionadas a um *chat*. Posto isto, é necessário compreender como é que cada uma dessas entidades se relacionam e qual a granularidade das associações das mesmas, ou seja, se a relação entre duas entidades é do tipo um para um (1:1), de um para muitos (1:m), ou se se trata de uma relação de muitos para muitos (m:n).

Uma vez identificados todo o tipo de dados que são necessários guardar e todas as relações de tabelas, passa-se à implementação do modelo de dados ou modelo entidade relação (ER). Através da Figura 17 é possível observar o modelo ER criado através da ferramenta *MySQL Workbench*.

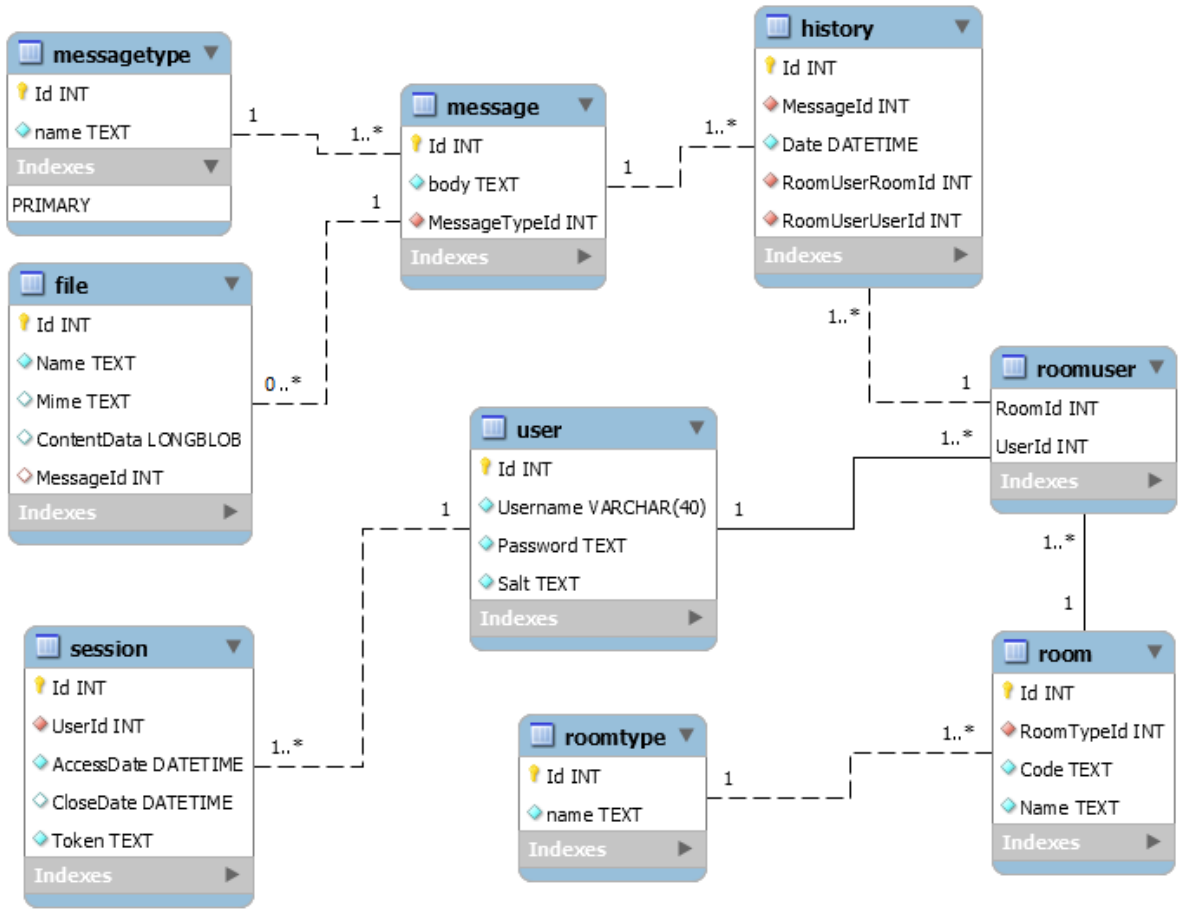


Figura 17: Modelos de Dados aplicacional (Modelo ER).

Tendo em conta o modelo de dados acima, pode compreender-se que se trata de uma estrutura de armazenamento de dados para uma aplicação de *chat*.

O modelo de dados implementado encontra-se preparado para a criação de salas de *chat* privadas, através da utilização de um '*roomType'* em específico. Sendo que através da relação entre a tabela '*room'* e '*roomType'* existe uma relação de *many-to-one* (m:1), ou seja, uma sala apenas pode ter um tipo, no entanto um tipo de sala pode representar várias salas. Seguindo o diagrama, é possível compreender que um utilizador pode estar ligado em diversas salas. Essa informação encontra-se presente na tabela '*roomuser'*, tratando-se de uma tabela m:n. A cada ligação estabelecida com a aplicação, após o sistema garantir a sua autorização, o mesmo autenticará o utilizador e criará uma entrada na tabela '*session'*.

Na tabela '*user'* constam todos os utilizadores do sistema e as suas credenciais, além disso, existe um processo de encriptação no sistema, em que a password é encriptada utilizando criptográfica a nível SHA256 Hash em conjunto com o valor da coluna *salt* também gerada no momento da criação do utilizador.

A tabela '*history'* permite identificar toda a informação necessária que é transmitida nas salas de *chat*. Através de uma data, de uma sala ou de um utilizador, é possível identificar todos os dados que são transferidos entre utilizadores. É, deste modo, na tabela '*message'* que está toda a informação enviada por um utilizador, numa dada ocasião, já que através da coluna *MessageTypeId* é possível identificar o tipo de dados enviados (texto ou ficheiro). Esta coluna trata-se de uma chave estrangeira para a tabela '*Messagetype'*.

Os ficheiros enviados são armazenados na tabela 'file', caso a mensagem corresponda a um ou mais ficheiros enviados, então a chave primária desta entrada na tabela *'message'* será utilizada como chave estrangeira na tabela '*file*' para cada um dos ficheiros enviados, representando, assim, a relação com a entrada da tabela '*message*'.

O modelo de dados acima descrito, permite que o sistema satisfaça o requisito de recuperação de dados perdidos entre sessões de um utilizador ou ligação.

4.2. Bibliotecas de Servidor baseado em *WebSockets*

Como já foi referido, foram escolhidas três bibliotecas de servidor para o desenvolvimento da solução, a qual servirá, mais tarde, para sustentar os testes de desempenho. As três bibliotecas foram selecionadas tendo em conta a linguagem de programação que seria utilizada. Além do que, a atividade das bibliotecas também constituiu um fator decisivo para a escolha das mesmas, uma vez que as bibliotecas sem suporte e sem atualizações não foram consideradas.

As três bibliotecas escolhidas neste projeto denominam-se *Fleck, WatsonWebSocket e vtortola WebSocketListener*. Qualquer das três bibliotecas suportam o protocolo de encriptação SSL.

A biblioteca *Fleck* [62] encontra-se na versão 1.1.0 trata-se de uma implementação servidor baseado em *WebSockets*, que está agregada a um *NuGet*. Este pacote não requer herança, *container* ou qualquer outra referência. O objetivo desta biblioteca é o desenvolvimento de uma solução que permita, de uma forma simples, iniciar a utilização de *WebSockets* em aplicações *web* .NET.

No caso da biblioteca *WatsonWebSocket* [63], esta encontra-se na versão 2.2.0.8. A finalidade da mesma assenta em encontrar a forma mais rápida e simples de construir uma aplicação cliente ou servidor, com comunicação via protocolo WS. Ademais, esta biblioteca também suporta o protocolo de encriptação SSL.

Por fim, a terceira biblioteca escolhida foi a biblioteca *vtortola WebSocketListener* [64] na versão 3.0.0. Esta é projetada para fornecer ligações *WebSockets* para outras aplicações. A mesma providencia métodos simples que permitem aceitar pedidos de ligações WS de forma assíncrona. Além disso, a biblioteca em questão suporta ligações seguras (WSS) e o envio de mensagens quer sejam do tipo binário ou de texto.

4.3. Solução de servidor baseado em *WebSockets* implementada

Durante todo o processo de desenvolvimento teve-se uma especial atenção para que a solução empregasse os princípios de desenho de *software* na programação orientada a objetos, garantindo assim as boas práticas de código e facilidade na perceção da implementação.

Desde as primeiras conceções da solução que se tem em atenção o respeito pelo princípio da responsabilidade única em qualquer desenvolvimento da mesma. Este princípio defende que cada classe deve ser responsável por uma única tarefa, realizando a sua função, desempenhando-a da melhor forma possível. Apesar de nos desenvolvimentos primordiais este princípio não ter sido sempre garantido, tenta-se, sempre que possível, realizar uma reformulação de modo a satisfazer o mesmo.

Considera-se que é adequado utilizar um padrão de desenho de criação no sistema: o padrão *Factory Method.* Este padrão define uma interface para a criação, sendo que a responsabilidade de decidir que classe será criada, ficará a cargo da subclasse. Este padrão é utilizado para a criação da biblioteca de servidor baseado em *WebSockets* segundo a opção escolhida, para iniciar o servidor com a biblioteca selecionada, através do menu de inicialização (*WebSocketServerFactory*). Para além do que, este padrão de desenho é utilizado para a criação de utilizadores (*UserFactory*) e a criação de sessões (*SessionFactory*). Ao iniciar a *Console Application* acede-se a um menu de entrada que permite a seleção da biblioteca, o que é possível verificar na Figura 18.

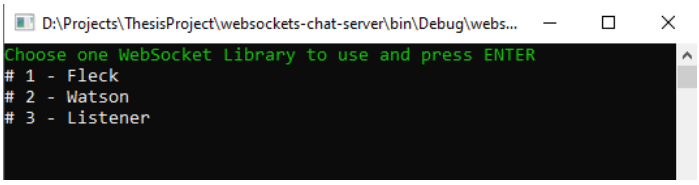


Figura 18: Menu de entrada da solução de consola implementada.

4.4. Aplicação chat para o canal *web*

Como forma de demostrar o desenvolvimento e funcionalidades da aplicação de servidor baseado em *WebSockets* durante este estudo, desenvolveu-se uma aplicação de *chat*. A vertente de *front-end* (FE) aplicacional é desenvolvida para a *web*, utilizando três tipos de linguagens: Linguagem de Marcação de Hipertexto (HTML), linguagem de programação JavaScript (JS) e linguagem *Cascading Style Sheets* (CSS). Segundo o *site Mozilla Developer Network*, a linguagem de programação JS é conhecida por ser uma linguagem de *scripting* para as páginas *web* e por ambientes como Node.js, *Apache CouchDB* e *Adobe Acrobat*. Além disso, é uma linguagem dinâmica, baseada em protótipos, *single-thread* e suporta vários estilos de programação (orientada a objetos, imperativa e declarativa) [65]. A estruturação da aplicação para o canal *web*, é definida através de HTML, com recurso a elementos da mesma linguagem, em que alguns destes encadeiam outros. Em termos de estilos e formatação de conteúdos, utiliza-se a linguagem CSS. Esta, é utilizada para descrever a apresentação de um documento escrito em HTML ou XML. A CSS constitui um mecanismo de adicionar estilos como tipos de letra, cores, espaçamentos e entre muitas outras opções, consoante a imaginação e habilidade do programador, tal como se pode observar no *site Mozilla Developer Network* para a linguagem CSS [66]. Observando a Figura 19, pode compreender-se que o HTML estrutura os textos, os botões, as listas e as imagens por trás das páginas *web*, já a CSS tem a responsabilidade de gerir a apresentação das páginas e o JS relaciona-se com a parte comportamental que pode alterar o HTML e o CSS.

Figura 19: Representação do conjunto padrão de desenvolvimento web (HTML + CSS + JS). [67]

Tendo por base o *site Mozilla Developer Network* para a API *WebSockets, a* aplicação *web* as ligações de *WebSockets* estabelecidas entre clientes e servidor são conseguidas

fazendo uso da *WebSockets* API [24]. Tal como já foi referido no tópico Transmissão de Dados através do Protocolo *WebSockets* da Literatura, esta API permite que se estabeleçam ligações de *WebSockets* entre o cliente e o servidor fazendo uso da linguagem *JavaScript*, como é visível na Figura 20.

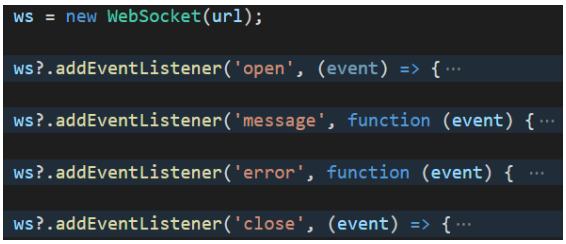


Figura 20: Inicialização do objeto WebSocket responsável pela ligação WS ou WSS.

A aplicação de *chat*, chama-se *WSWebChat*, e é desenvolvida em domínio local. Todos os dados são armazenados numa DB Relacional, *MySQL*. Observando a Figura 21 compreende-se que, só será possível usufruir da aplicação, se o utilizador efetuar uma autenticação, aquando do acesso à mesma, utilizando um *username* e uma *password*, com um mínimo de 8 carateres.

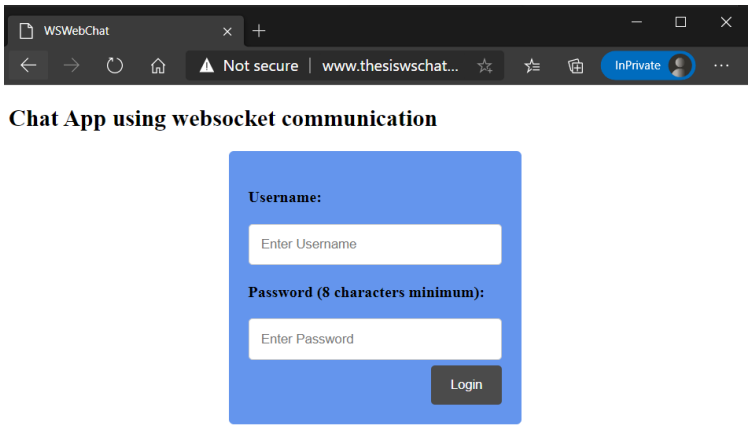


Figura 21: Ecrã de autenticação da aplicação de chat WSWebChat.

Após o momento de *login*, o utilizador acederá à área privada que permitirá que este disfrute de algumas funcionalidades interessantes, como a criação de novas salas de diálogo, a possibilidade de submissão de anexos e ainda as mensagens de texto. Existem ainda outras duas caraterísticas que tornam esta aplicação relativamente interessante, estando estas relacionadas com as notificações de novas mensagens recebidas em outras salas de *chat* e a recuperação de mensagens perdidas desde o encerramento da última sessão ou ligação.

Um utilizador, ao realizar *login* entrará por defeito numa sala padrão a todos os utilizadores, chamada "*Lobby*". Esta não possui a funcionalidade de histórico, nem permite que o utilizador se auto exclua da sala, uma vez que a ideia aplicacional de negócio requer que exista uma sala comum a todos os utilizadores, caso seja pretendido transmitir algum dado ou informação a todos, garantindo assim que os mesmos sejam notificados. A Figura 22 serve para exemplificar o funcionamento de troca de mensagens entre dois utilizadores, além disso, é possível observar o aspeto visual da aplicação e das funcionalidades que o utilizador pode usufruir. Consultando o anexo em 8.1 pode visualizar-se outras mais figuras que demonstram, em mais detalhe, a aplicação.

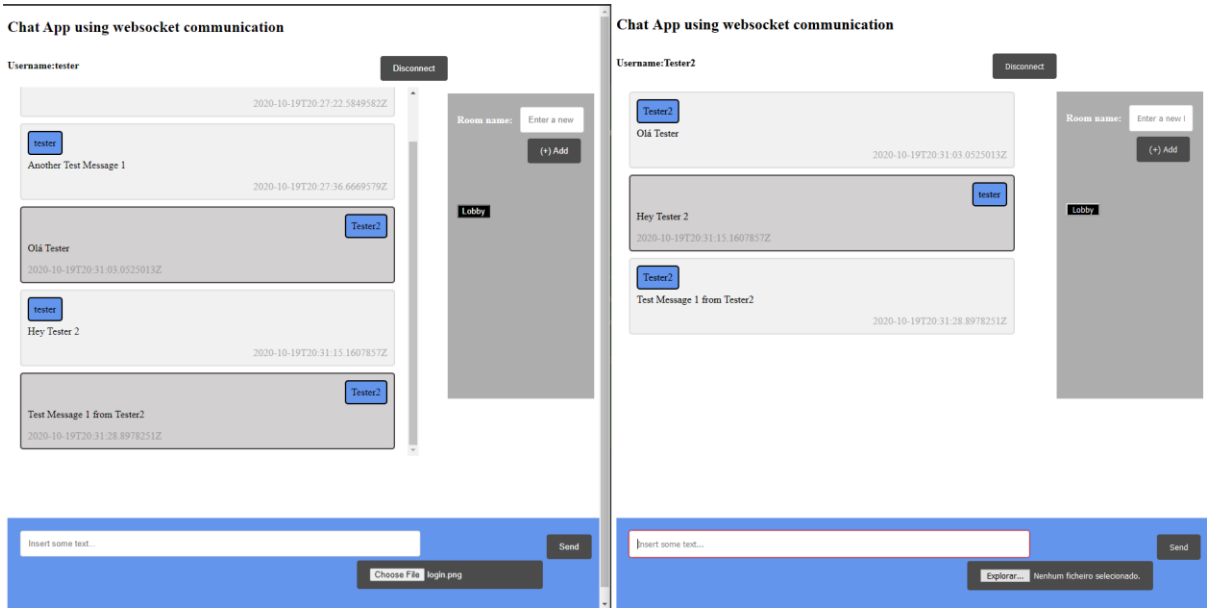


Figura 22: Exemplificação de troca de mensagens entre dois utilizadores.

4.5. Seleção das ferramentas de testes

Nesta secção, pretende-se descrever a seleção das duas ferramentas de testes de desempenho, além disso, serão também descritas as diferentes provas efetuadas sob as diferentes bibliotecas e ferramentas selecionadas, que servirão para validar o cumprimento dos objetivos da solução estipulados em 1.4. Numa primeira etapa será explicado todo o processo que levou à seleção das duas ferramentas. Posto isto, demonstrar-se-á que tipo de testes foram realizados, descrever-se-á cada um dos cenários de teste, apresentar-se-á para cada ferramenta, o resultado da criação de um cenário, o local de armazenamento dos dados provenientes dos testes e por fim referir-se-á o ambiente em que serão realizados os testes.

A seleção destas ferramentas consiste num momento de identificação dos requisitos que a ferramenta deve satisfazer. A condição principal para a seleção das ferramentas tem por base o suporte para o protocolo WS. Existem outras especificações que suportam a escolha das ferramentas tais como a permissão de simular múltiplas ligações concorrentes, a recolha de indicadores chave de desempenho (*Key Performance Indicator* – KPI), nomeadamente, os tempos de resposta, a latência, o máximo de utilizadores virtuais (*Virtual Users* – VUs) concorrentes e o (*throughput*) e, por fim, retomando as especificações, a criação automática de relatórios. Além disso, existem ainda outros fatores importantes no momento da escolha de uma ferramenta de testes: o tipo *Hardware/Software* que a ferramenta de automação requer, bem como, o suporte e as políticas de atualizações da ferramenta.

Estas ferramentas auxiliam na realização dos testes ao desempenho do *software*. Para Kaur e Gupta, existem muitas ferramentas de testes de *software* e de desempenho disponíveis, quer comerciais ou *Open Source*. Uma parte das ferramentas não comercializadas que se encontram disponíveis no mercado de *software*, são especializadas para a realização de testes *web* ou de aplicações nativas. Contudo, as principais funcionalidades desse nicho de ferramentas são semelhantes, diferindo apenas em algumas funcionalidades, caraterísticas ou em termos de usabilidade [54].

Para o correto manuseamento de uma ferramenta deve estudar-se a documentação existente, já que existem muitos projetos de testes de desempenho que se deparam com

dificuldades, no decorrer da fase de *scripting*, devido à insuficiente avaliação técnica da ferramenta, conforme menciona Hooda e Rajender [43]. Deste modo, as ferramentas de testes de desempenho que se seguem foram selecionadas como resultado de um procedimento prévio de execução de uma POC a cada ferramenta recolhida e da filtragem consoante os requisitos estabelecidos.

De todas as ferramentas que se podem encontrar no mercado, para este estudo, selecionaram-se duas: a *Apache Jmeter* (versão 5.3) e a *NeoLoad* (versão 7.6). A comparação das mesmas encontra-se reunida na Tabela 6.

Tendo em conta *o site* relativo à ferramenta *Apache JMeter,* esta é uma ferramenta *Open Source* cuja linguagem utilizada é a Java. Foi desenhada especialmente para testar o comportamento funcional de carga e medir o desempenho de um sistema. Esta ferramenta foi originalmente projetada para que se pudessem realizar testes em aplicações web, no entanto, desde a sua criação que se difundiu para outras funções de teste [68].

A ferramenta em questão, de acordo com a fonte supramencionada, pode ser utilizada para testar o desempenho em recursos estáticos ou dinâmicos, daí a sua versatilidade, razão pela qual é uma das selecionadas para integrar este estudo. Além disso, a mesma pode ser usada para fins de simulação de cargas de um grupo de servidores ou de um servidor em específico, de uma rede ou mesmo de um objeto, para testar a sua resistência ou analisar o seu desempenho geral sob diferentes tipos de carga. Ademais, o facto de existir a possibilidade de se integrarem *plug-ins* desenvolvidos em Java, permite um ilimitado número de recursos de testes [68].

Conforme refere Mallala, a *NeoLoad* é uma ferramenta comercial altamente eficiente para testes de carga e de stress, de baixo custo, utilizada para medir o desempenho de aplicações *web* e *mobile* [69]. Esta ferramenta simula o tráfego através de VUs para determinar o desempenho aplicacional sobre carga e monitoriza métricas de *performance*. Além disso trata-se de uma ferramenta com uma GUI muito *user-friendly.* A mesma suporta diversas tecnologias e protocolos da *web* e permite gerar melhores relatórios de teste, tal como é possível consultar no *site* referente à ferramenta em questão [70].

Uma das razões pelas quais esta ferramenta é uma das selecionadas para este estudo, prende-se com o facto de a sua utilização ser simples. De acordo com o *site* citado anteriormente, existem várias funcionalidades que simplificam a criação do *design* de teste, como ciclos, condições de verificação através de funcionalidades de arrastar e soltar (*drag and drop*). Apesar da simplicidade desta ferramenta, a mesma pode ser utilizada para casos extremamente avançados. Porém, não é necessário que se seja especialista para utilizar a NeoLoad para testar um sistema. Pode considerar-se, portanto, uma ferramenta muito versátil [70].

Ainda segundo o *site* referente à ferramenta *NeoLoad*, outra das vantagens que esta ferramenta apresenta é o facto de assentar numa forma mais rápida de correlacionar parâmetros dinâmicos com regras predefinidas para as *frameworks* mais comuns. Além disso, a presente ferramenta converte *scripts* de testes funcionais em *scripts* de testes de desempenho com a finalidade de acelerar o *design* e a manutenção do mesmo. Por fim, a *NeoLoad* permite que se atualize um *script* de teste com a informação resultante de uma nova gravação, mantendo de forma automática caraterísticas anteriormente definidas tais como:

extratores de variáveis, tempos de reflexão, ciclos e validações. Comparativamente a outras ferramentas deste género, esta é uma das que se encontra mais bem documentada, ajudando assim à sua utilização [70].

Todas as ferramentas anteriormente descritas foram selecionadas para integrar o presente estudo tendo em conta o facto de todas suportarem o protocolo de comunicação WS.

Critérios	Subcritérios	Apache JMeter 5.3	NeoLoad 7.6
Custo	Licença comercial	Não (Gratuita)	Sim
	Pacote base	Gratuita independentemente do número de utilizadores	Gratuita até 50 Utilizadores Virtuais. A licença depende do número de utilizadores e outras informações
	Extras adicionais	Não aplicável	Pagamento por cada customização, por cada suporte protocolar e monitorização adicional
Funcionalidades	Suporta WS	Sim (através da instalação de plugins)	Sim
	Suporta gravação do browser	Sim	Sim
	Geração de Relatórios	Sim	Sim
	Geração de Carga	Ilimitada	Depende do tipo de Licença
	Configuração de testes de carga via código	Através da ferramenta Taurus	Sim
	Customização de *Scripts*	Simples	Medio/Difícil (JS para cenários complexos)

Tabela 6: Resumo das ferramentas selecionadas.

Para a identificação da biblioteca com melhor desempenho, estabeleceu-se que será utilizada a técnica de testes não funcionais: testes de desempenho do tipo testes de carga, para cada uma das bibliotecas.

Os testes de desempenho são realizados, como já foi referido, fazendo uso de duas ferramentas de testes, a ferramenta *Apache JMeter* e a *NeoLoad*. A utilização de mais que uma ferramenta, permite ter uma melhor perceção dos relatórios obtidos, bem como verificar a existência de alguma variação nos resultados ao permutar de ferramenta.

Para uma melhor adaptação ao mundo real, define-se quatro cenários de teste: o primeiro cenário trata-se de um ambiente com um número reduzido de 10 utilizadores ativos concomitantemente procedendo ao envio de 20 mensagens de texto, o que permite simular, desta forma, um sistema em períodos de pouca utilização; o segundo cenário é constituído com o mesmo número de utilizadores ativos, porém com um aumento do número de mensagens enviadas para 500 por utilizador; o terceiro trata-se de um cenário que conta com 50 utilizadores e 20 mensagens e o quarto e último cenário é constituído por 50 utilizadores e 500 mensagens. A Tabela 7 resume os cenários utilizados para a execução de testes.

Cenário	Utilizadores	Mensagens
10 VUs 20 Mensagens	10	20
10 VUs 500 Mensagens	10	500
50 VUs 20 Mensagens	50	20
50 VUs 500 Mensagens	50	500

Tabela 7: Resumo de cenários de teste.

A estruturação dos *scripts* para os cenários acima estipulados é realizada de forma minuciosa, tendo em atenção uma tentativa de aproximação dos cenários de teste ao real, de forma que os resultados obtidos permitam tecer conclusões profícuas.

Para isso, em cada uma das ferramentas, são definidos tempos de pausa (*think times*) após o estabelecimento das ligações de *WebSockets* e após o envio de uma mensagem. Um tempo de pausa nos testes de desempenho tem como funcionalidade simular a diferença de tempo entre cada ação efetuada por um utilizador, este ao utilizar uma aplicação *web*, nativa ou *mobile* despende de algum tempo, antes de fazer qualquer ação. Além disso, para qualquer um dos cenários as mensagens simuladas têm um comprimento aleatório e tamanho máximo de 627 *bytes*.

Para as ferramentas selecionadas, os cenários de teste são desenvolvidos utilizando as *GUIs* que estas possuem. A ferramenta *Apache JMeter* detém a organização de ficheiros presente na Figura 23.

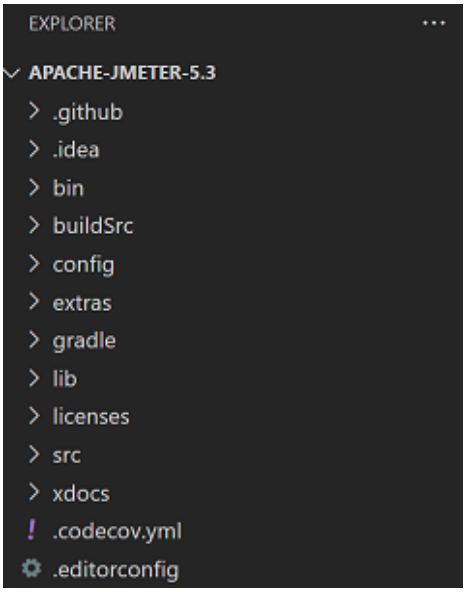


Figura 23: Estrutura de ficheiros da ferramenta Apache JMeter.

Para iniciar esta ferramenta, o utilizador deve encaminhar-se ao diretório *bin* e executar o ficheiro jmeter.bat (em *Windows*) ou jmeter.sh (em *Linux*), que iniciará a ferramenta.

Para criação do cenário de teste nesta ou em qualquer outra ferramenta deve, em primeiro lugar, consultar-se a documentação da mesma. A Figura 24 reflete o resultado da criação de um cenário de teste para esta ferramenta. As ferramentas que permitem definir os cenários de teste através de uma GUI, possibilitam que outras pessoas além dos *testers* ou dos programadores consigam preparar os cenários de teste.

Além disso, no que concerne a alterações do cenário de teste, a utilização de uma GUI, torna-se uma mais valia pela sua facilidade de modificação e de acesso.

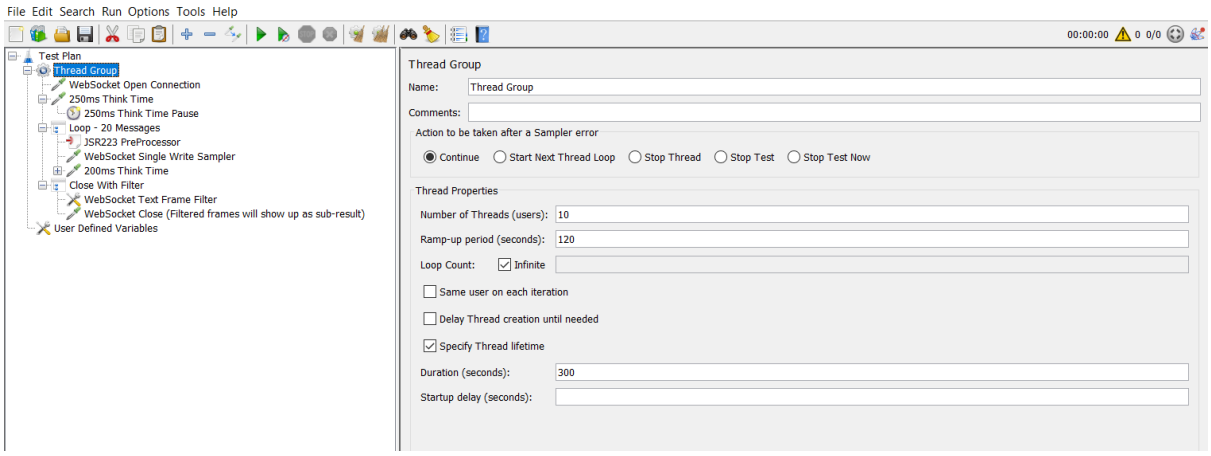


Figura 24: Exemplo da definição de um cenário para a ferramenta Apache JMeter.

A ferramenta *NeoLoad* possui uma estruturação um pouco distinta, já que por defeito utiliza o diretório */Documents/NeoLoad Projects/*v7.6 com uma pasta para cada projeto, em vez do diretório de instalação para guardar os projetos nela criados ao contrário de outras ferramentas. O projeto desenvolvido nesta ferramenta designa-se *WSLibrariesPerformanceTest* e foi criada uma pasta com o nome do mesmo, como se pode verificar na Figura 25.

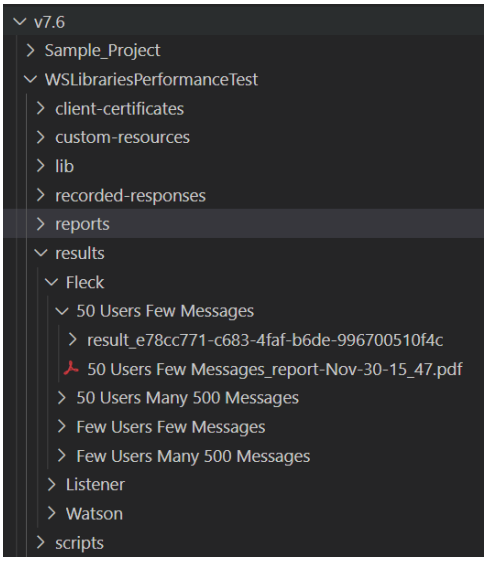


Figura 25: Estrutura de ficheiros da ferramenta NeoLoad.

Para iniciar esta ferramenta, deve executar-se o ficheiro *NeoLoadGUI*.exe e em seguida criar um projeto. Esta ferramenta tem uma vantagem, em comparação com a *JMeter*, pois maioria das funcionalidades para montar um cenário de teste encontram-se visíveis ao primeiro contacto com a GUI no canto inferior esquerdo. No entanto, esta ferramenta apresenta a possibilidade de o utilizador desenvolver as suas funcionalidades de forma customizada, através de uma 'ação', utilizando a linguagem *JavaScript*.

Na Figura 26 pode consultar-se o resultado da definição do cenário de teste para simular 10 VUs e 500 Mensagens.

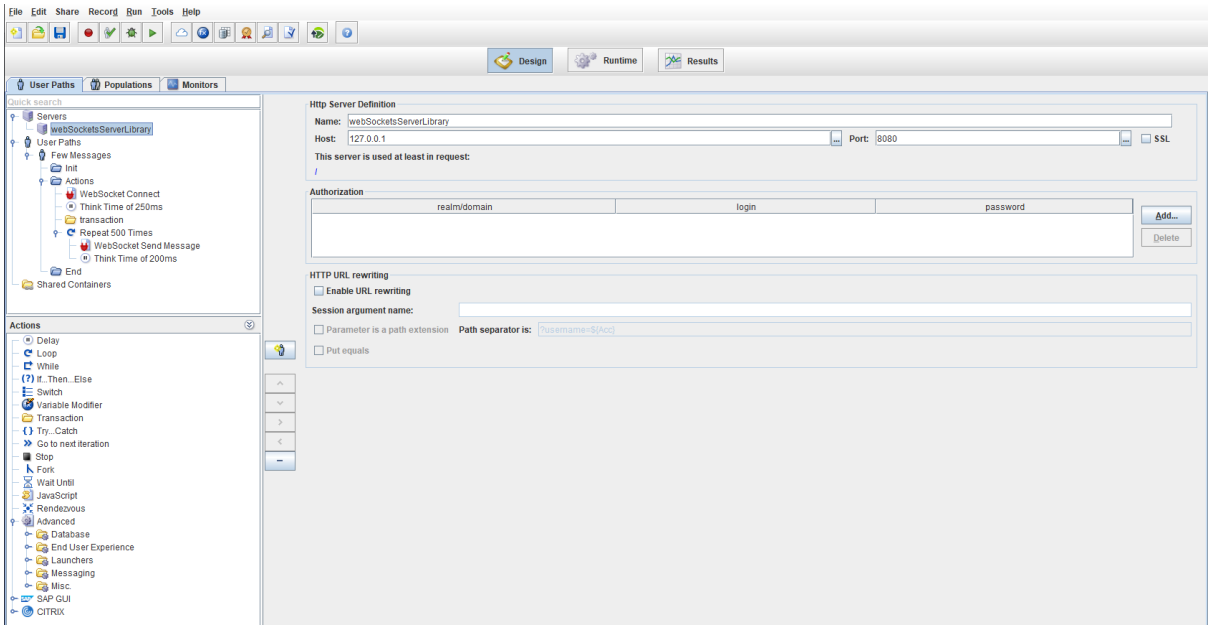


Figura 26: Exemplificação da definição de um cenário para a ferramenta NeoLoad.

O ambiente no qual são realizados os testes de carga, trata-se de uma máquina local que possui as caraterísticas presentes na Tabela 8. Esta máquina serve para executar os testes de carga, como também para desempenhar o papel de servidor. Tal como já foi referido em 4.3, trata-se de uma solução de servidor do tipo de projeto *Console Application* em ASP.Net *Framework*.

Componente	Nome
Processador (CPU)	Intel Core i7 7700HQ 4 Cores
Memória RAM	24Gb RAM
Sistema Operativo	Windows 10 *Education* x64

Tabela 8: Caraterísticas do ambiente de testes.

5. Resultados

Neste tópico serão apresentados os resultados dos testes efetuados nas ferramentas selecionadas para cada uma das bibliotecas. Além disso, pretende-se realizar um momento de discussão acerca dos dados recolhidos nos diferentes relatórios, resultantes da execução dos testes para os diferentes cenários. Esta discussão tem como finalidade responder ao à questão de pesquisa estabelecida em 1.3 e ainda verificar se os objetivos determinados em 1.4 foram cumpridos.

Todos os testes são realizados na máquina que possui o servidor em execução. Cada biblioteca de servidor é testada para cada um dos quatro cenários estabelecidos com uma duração de cinco minutos. O arranque do teste leva dois minutos até que se alcance o número de VUs necessários a cada cenário e após esse momento, o número de VUs ativos mantém-se até ao final do teste.

Para uma melhor perceção dos dados recolhidos, reuniram-se os resultados de cada biblioteca em duas tabelas, cada uma destas referindo-se a uma ferramenta em específico. Ao observar-se cada uma das tabelas separadamente, pode verificar-se a variação de valores recolhidos para cada biblioteca, em cada um dos diferentes cenários. Como já foi referido, o objetivo é verificar que biblioteca de servidor baseado em *WebSockets* se destaca em termos de melhor desempenho. Este parâmetro foi medido através de métricas resultantes dos testes de carga tais como: número de utilizadores ativos, duração dos testes, percentagem de erros, memória, CPU, *throughput* e quantidade de dados enviados e recebidos.

Os dados recolhidos dos testes de carga executados pela ferramenta *Apache JMeter* podem observar-se na Tabela 9. Através dos dados obtidos nesta ferramenta, é possível observar que *Listener* apresentou valores significativamente mais baixos nas métricas referentes às amostras, transações por segundo e dados recebidos. Além disso, é nesta biblioteca que se verificam taxas de erro mais altas, principalmente para o cenário (c), que é, consequentemente, o cenário no qual se verifica uma maior diferença dos dados relativamente às outras bibliotecas para os parâmetros anteriormente referidos.

Deste modo, conclui-se que, segundo esta ferramenta, a biblioteca que apresenta um melhor desempenho é a biblioteca *Fleck*, uma vez que é nesta que se verifica uma menor taxa de erros aquando da maior carga.

Métricas \ Biblioteca		Fleck	Watson	Listener	Cenário
Execução	Amostras	14532	89682	12835	(a)
		97401	97401	13026	(b)
		2567316	2563903	215586	(c)
		1958664	97778	71639	(d)
	Erros (%)	0	0	0.15	(a)
		0	0	0.15	(b)
		0	0	1.27	(c)
		0.07	0.09	0.13	(d)
Throughput	Transações/ sec. (KB/ sec.)	324.45	298.99	42.79	(a)
		324.69	324.74	43.43	(b)
		8557.58	8546.23	718.64	(c)
		8694.50	325.94	238.80	(d)
Rede	Dados Recebidos (KB/ sec.)	88.34	80.43	0.40	(a)
		89.03	87.97	0.48	(b)
		2589.54	2597.05	80.57	(c)
		2617.48	39.04	6.52	(d)
	Dados Enviados (KB/ sec.)	12.81	12.64	12.75	(a)
		12.73	12.61	12.64	(b)
		59.71	60.01	59.68	(c)
		77.41	62.02	62.02	(d)

Cenários:

(a) – 10VUs - 20 Mensagens
(b) – 10VUs - 500 Mensagens
(c) – 50VUs - 20 Mensagens
(d) – 50VUs - 500 Mensagens

Tabela 9: Resultados dos testes de carga para a ferramenta Apache JMeter.

Os dados recolhidos dos testes de carga executados para a ferramenta *NeoLoad* foram reunidos na Tabela 10. Aquando da análise do relatório verifica-se um valor atípico para a biblioteca *Fleck*, nomeadamente para o parâmetro *throughput*. Enquanto as bibliotecas *Watson* e *Listener* apresentam um aumento do número de transações por segundo a cada cenário, a biblioteca *Fleck* apresenta um decréscimo deste número no cenário (d), perfazendo uma diferença de 1260 transações por segundo relativamente ao cenário (c). É de referir que, além do que se explanou anteriormente, verifica-se que a biblioteca *Fleck*, quando comparada com a Watson e Listener, apresenta mais transações por segundo nos cenários (a), (b) e (c), porém no cenário (d) observa-se o oposto.

Assim, apurou-se que a biblioteca *Fleck* não apresenta um bom desempenho quando a carga é mais elevada.

Esta ferramenta permite, ainda, monitorizar outras métricas, que não estão referidas na tabela uma vez que as restantes ferramentas não o permitem, não existindo assim um ponto de comparação, nomeadamente, o estado do servidor que é submetido às cargas estabelecidas em cada cenário e que se reflete na percentagem de CPU utilizada.

É importante referir que, aquando do teste, esta ferramenta registou vários alertas de elevada utilização de processador para dois cenários da biblioteca *Fleck*: o cenário (c) e o cenário (d) que simularam 50 VUs e 20 mensagens enviadas por cada um destes e 50 VUs e 500 mensagens, respetivamente. Esses alertas ocorreram para o cenário (c), com uma utilização de CPU acima dos 90%, e para o cenário (d) duas ocorrências com uma utilização de CPU acima de 80% e três ocorrências com utilização de CPU acima dos 90%.

Para o servidor Watson, verifica-se que existiram alertas de elevada utilização de processador (CPU) acima dos 80% para os dois cenários (c) e (d), tendo sido verificadas três ocorrências para o cenário (c) e apenas uma para o cenário (d).

Por último, observa-se que para o servidor *Listener*, existiram alertas de elevada utilização de CPU para os cenários (c) e (d), sendo que em ambos apenas existiu um alerta de utilização de CPU acima dos 80%.

Métricas \ Biblioteca		Fleck	Watson	Listener	Cenário
Execução	Amostras	112499	112409	97803	(a)
		122009	122212	103184	(b)
		1079809	1034484	868099	(c)
		1069502	1082128	1088153	(d)
	Erros (%)	0	0	0	(a)
		0	0	0	(b)
		0	0	0	(c)
		0	0	0	(d)
Throughput	Transações/ sec. (KB/ sec.)	820	830	700	(a)
		910	900	730	(b)
		8440	8050	6690	(c)
		7180	8420	8410	(d)
Rede	Dados Recebidos (KB/ sec.)	17	10	16	(a)
		109	9	16	(b)
		13	31	203	(c)
		56	21	24	(d)
	Dados Enviados (KB/ sec.)	4	2	3	(a)
		207	3	18	(b)
		9	8	8	(c)
		305	16	39	(d)

Cenários:

(a) – 10VUs - 20 Mensagens
(b) – 10VUs - 500 Mensagens
(c) – 50VUs - 20 Mensagens
(d) – 50VUs - 500 Mensagens

Tabela 10: Resultados dos testes de carga para a ferramenta NeoLoad.

Assim, ao comparar, primeiramente as duas ferramentas, pode concluir-se que, os relatórios gerados pela *NeoLoad* verificam-se mais completos, uma vez que apresentam métricas que não se observam nos gerados pela *Apache JMeter*. Deste modo, os dados resultantes daquela são mais esclarecedores para dar resposta à questão de pesquisa. Da primeira ferramenta que se analisou, a *Apache JMeter*, ordena-se de melhor para pior desempenho as bibliotecas, da seguinte forma: *Fleck*, *Watson* e *Listener*. Já em relação à segunda ferramenta utilizada, a *NeoLoad*, a ordenação, segundo o mesmo critério, é a seguinte: *Watson*, *Listener* e *Fleck.*

Desta maneira e respondendo à questão de pesquisa enunciada em 1.3, sendo que a biblioteca *Watson* não é considerada como a de pior desempenho em ambos os testes, conclui-se que a biblioteca que apresenta a melhor performance é a Watson.

Em relação ao primeiro objetivo, que assenta em desenvolver uma solução que permita executar testes de desempenho, considera-se que este tenha sido cumprido. Foi elaborado um servidor em C# e em ASP.NET *framework* que possibilitou que se realizassem testes de desempenho.

No que respeita ao segundo objetivo estabelecido, sendo este a pretensão de se testarem todas as bibliotecas sob diferentes condições de utilização, também foi cumprido.

As bibliotecas foram submetidas a testes de carga com recurso a duas ferramentas e em diferentes cenários que correspondiam a condições de utilização distintas como é possível observar na Tabela 7.

No que concerne ao terceiro objetivo enunciado, que se prende com a identificação da biblioteca com melhores resultados, também se verifica ter sido alcançado. Observou-se que, após a análise dos relatórios dos testes de carga gerados pelas ferramentas selecionadas, se conseguiu concluir que a biblioteca com melhor desempenho era a biblioteca *Watson*.

Por fim, relativamente ao quarto e último objetivo deste projeto, sendo este a utilização da biblioteca com o melhor desempenho para desenvolver um servidor baseado em *WebSockets* para os projetos das empresas clientes da TEKEVER, trata-se do único objetivo que não foi cumprido. O não cumprimento deste objetivo está relacionado com a situação que a empresa atravessava na altura em que ocorreu o estágio, em que foi proposto ao estagiário integrar outros projetos na empresa, acabando por influenciar a gestão do tempo deste projeto em específico, o que consistiu num dos maiores constrangimentos deste projeto.

6. Conclusões

Com o culminar desta etapa é imprescindível que se reflita sobre todo o percurso realizado ao longo de todo o curso. No ano letivo 2018/2019 foi realizado um estágio, no âmbito na unidade curricular Projeto / Estágio / Dissertação que se considera essencial à formação e que é agora concluída.

O estágio é na verdade uma experiência que permite ao estagiário, na prática, assimilar os conhecimentos teóricos adquiridos outrora. Segundo Zarifian citado por Francisco, um estágio além de transmitir conhecimentos, também transforma o estagiário num ser reflexivo sobre si mesmo e sobre o ambiente que o rodeia, ou seja, um estágio serve para esculpir as melhores capacidades do estagiário sejam elas individuais ou sociais e com elas produzir novas competências [71].

Este estágio representa um enorme contributo para o percurso profissional que já foi iniciado. Além disso, o contributo tanto das pessoas que orientaram o estagiário como do professor orientador foi deveras fundamental para o bom funcionamento do estágio. É de referir que foi muito importante todo o suporte facultado por estes perante todas as tomadas de decisão.

Através deste estágio proporcionou-se a realização deste estudo, pelo que o desenvolvimento do mesmo permitiu estudar e aplicar testes de automação para a técnica não-funcional relativamente aos testes de desempenho, mais precisamente os testes de carga. Além disso, este estudo possibilitou o contacto com as ferramentas que permitiram a realização dos testes.

Ao nível deste trabalho em específico, foi desenvolvida uma solução que permitiu a seleção da biblioteca servidor baseado em *WebSockets* através de um menu construído na solução do tipo *Console Application*.

No que concerne às dificuldades sentidas neste projeto, o maior constrangimento esteve em torno dos testes de desempenho, mais propriamente na criação dos *scripts* de teste para cada ferramenta. Uma outra dificuldade que se refletiu nos resultados esteve relacionada com a seleção de ferramentas compatíveis com o protocolo de comunicação de *WebSockets* e que fossem *Open Source*, apesar de se considerar que as ferramentas comerciais deram também um excelente contributo ao estudo.

Ademais, devido ao facto do desenvolvimento da solução, necessária à realização dos testes, ter sido concebida utilizando a linguagem C# com recurso a ASP.NET *Framework* da *Microsoft*, fez todo o sentido que se desenvolvesse uma solução utilizando a tecnologia *SignalR*. No entanto, devido à estrutura da solução estabelecida para esta tecnologia e à gestão do tempo, não foi possível a integração desta solução no projeto. As ferramentas utilizadas não suportam a realização de testes de carga para esta tecnologia, pelo que não existiria meio de comparação entre as tecnologias para as diferentes ferramentas, dificultando, assim, a seleção da biblioteca com melhor desempenho.

Considera-se que a finalização deste estudo se verifica muito enriquecedora tanto a nível pessoal como profissional. Além disso, é expectável que constitua, também, uma mais-valia para a empresa TEKEVER, pois passará a ter documentado todo um estudo em torno

desta tecnologia, resultando, deste modo, numa nova *feature* que poderá ser integrada nos seus projetos *on-going* e futuros.

Relativamente a trabalhos futuros, é importante que se refira que a primeira solução nunca será a final, existem sempre componentes do projeto que precisam de ser melhoradas, como também a integração e interligação de novas funcionalidades. No que respeita à solução de servidor pretende-se melhorar aspetos que precisem de uma factorização, já que a experiência e conhecimento serão aperfeiçoados. Uma outra funcionalidade que poderá ser integrada no futuro relaciona-se com a possibilidade de o servidor possuir a hipótese de enviar dados em *bytes*.

Além disso, a eventualidade de se alertar a equipa de programadores sobre falhas através de *email*, melhorará a capacidade de detetar problemas com maior facilidade, já que estes alertas permitem que se replique um mesmo problema e assim se consiga identificar o que está menos bem. Como última melhoria, seria interessante efetuar novos conjuntos de simulações baseadas em cenários de teste com maior detalhe em termos de ações efetuadas por utilizadores reais.

Em suma, todas as tarefas que são efetivadas com dedicação e determinação resumem-se em algo primoroso e com potencial, como referem DeMarco e Lister, através da seguinte citação: *"Quality is free, but only to those who are willing to pay heavily for it"* [72].

7. Referências

[1] N. Pandey and D. Bein, 'Web application for social networking using RTC', in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference, CCWC 2018*, Feb. 2018, vol. 2018-Janua, pp. 336–340, doi: 10.1109/CCWC.2018.8301692.

[2] Y. Zhangling and D. Mao, 'A Real-Time Group Communication Architecture Based on WebSocket', *Int. J. Comput. Commun. Eng.*, vol. 1, no. 4, pp. 408–411, 2012, doi: 10.7763/ijcce.2012.v1.100.

[3] I. Hickson, 'The WebSocket API', *W3C website*, Sep. 20, 2012. https://www.w3.org/TR/2012/CR-websockets-20120920/ (accessed Jun. 21, 2020).

[4] E. Estep, 'Mobile HTML5: Efficiency and Performance of WebSockets and Server-Sent Events', 2013. Accessed: Jun. 21, 2020. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-177872.

[5] Matt West, 'An Introduction to WebSockets | Treehouse Blog', Oct. 2013. https://blog.teamtreehouse.com/an-introduction-to-websockets (accessed Jun. 21, 2020).

[6] I. Hickson, 'WebRTC 1.0: Real-time Communication Between Browsers', *W3C website*, 2019. https://www.w3.org/TR/webrtc/ (accessed Jun. 21, 2020).

[7] 'Server-sent events - Wikipedia'. https://en.wikipedia.org/wiki/Server-sent_events (accessed Jun. 21, 2020).

[8] I. Hickson, 'Server-Sent Events', *W3C website*, 2015. https://www.w3.org/TR/2015/REC-eventsource-20150203/ (accessed Jun. 21, 2020).

[9] Michael Carter, 'TCPConnection feedback', Jun. 17, 2008. https://lists.w3.org/Archives/Public/public-whatwg-archive/2008Jun/0165.html (accessed Jun. 21, 2020).

[10] 'IRC logs: freenode', Jun. 22, 2018. https://krijnhoetmer.nl/irc-logs/whatwg/20080618#l-1145 (accessed Jun. 21, 2020).

[11] and T. Y. Yuzo Fujishima, Fumitoshi Ukai, 'Chromium Blog: Web Sockets Now Available In Google Chrome', 2009. https://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html (accessed Jun. 21, 2020).

[12] M. González-Fierro, 'Demystifying WebSockets for Real-Time Web Communication', 2018. https://miguelgfierro.com/blog/2018/demystifying-websockets-for-real-time-web-communication/ (accessed Jun. 23, 2020).

[13] A. Marchetto, P. Tonella, and F. Ricca, 'State-based testing of Ajax Web applications', in *Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008*, 2008, pp. 121–130, doi: 10.1109/ICST.2008.22.

[14] C. A. Gutwin, M. Lippold, and T. C. N. Graham, 'Real-time groupware in the browser : Testing the performance of web-based networking', in *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*, 2011, pp. 167–176, doi: 10.1145/1958824.1958850.

[15] A. I. Nadaf, S. V Kulkarni, P. P. Shaha, and M. K. Bhanarkar, 'Review Paper on AJAX Comet and Websocket Uses for Web HMI/SCADA', *Int. J. Eng. Res. Gen. Sci.*, vol. 3, no. 5, 2015, Accessed: Jun. 23, 2020. [Online]. Available: www.ijergs.org.

[16] L.-J. Cui, H. He, and H.-W. Xuan, 'Analysis and Implementation of an Ajax-enabled Web Crawler', *Int. J. Futur. Gener. Commun. Netw.*, vol. 6, no. 2, pp. 139–146, 2013,

Accessed: Jun. 23, 2020. [Online]. Available: https://www.earticle.net/Article/A207961.

[17] L. Peter, G. Frank, and C. Kaazing, ‘HTML5 WebSocket - A Quantum Leap in Scalability for the Web’. http://www.websocket.org/quantum.html (accessed Jun. 23, 2020).

[18] E. van Rees, ‘Introduction to WebSockets’, *Scaledrone website*, 2018. https://www.scaledrone.com/blog/introduction-to-websockets/ (accessed Jun. 23, 2020).

[19] S. Loreto, P. Saint-Andre, S. Salsano, and G. Wilkins, ‘Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP’, Apr. 2011. doi: 10.17487/rfc6202.

[20] V. Pimentel and B. G. Nickerson, ‘Communicating and displaying real-time data with WebSocket’, *IEEE Internet Comput.*, vol. 16, no. 4, pp. 45–53, 2012, doi: 10.1109/MIC.2012.64.

[21] ‘WebSocket - Wikiwand’. http://www.wikiwand.com/en/WebSocket (accessed Jun. 23, 2020).

[22] L. Deveria, Alexis; Schoors, ‘Can I use... Support tables for HTML5, CSS3, etc’. https://caniuse.com/#feat=websockets (accessed Jun. 25, 2020).

[23] I. Hickson, S. Pieters, A. Kesteren, P. Jägenstedt, and D. Denicola, ‘HTML Standard’. https://html.spec.whatwg.org/#network (accessed Jun. 25, 2020).

[24] ‘The WebSocket API (WebSockets) - Web APIs | MDN’, *Mozilla Firefox developer website*. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (accessed Jun. 25, 2020).

[25] Y. Yemini, ‘The OSI network management model’, *IEEE Commun. Surv. Tutorials*, vol. 3, no. 1, pp. 20–29, Dec. 2009, doi: 10.1109/comst.2000.5340720.

[26] I. Fette and A. Melnikov, ‘The WebSocket Protocol’, Dec. 2011. Accessed: Jul. 14, 2020. [Online]. Available: https://tools.ietf.org/html/rfc6455.

[27] A. Freier, P. Karlton, and P. Kocher, ‘The Secure Sockets Layer (SSL) Protocol Version 3.0’, 2011. Accessed: Jul. 14, 2020. [Online]. Available: https://tools.ietf.org/html/rfc6101.

[28] R. Barnes, M. Thomson, A. Pironti, and A. Langley, ‘Deprecating Secure Sockets Layer Version 3.0’, May 2015. Accessed: Jul. 13, 2020. [Online]. Available: https://tools.ietf.org/html/rfc7568.

[29] ‘What are Web Sockets?. Web socket real time apps are more… | by Dominik Tarnowski | Medium’. https://medium.com/@td0m/what-are-web-sockets-what-about-rest-apis-b9c15fd72aac (accessed Jul. 27, 2020).

[30] F. S. P. M. Vanessa Wang, *The Definitive Guide to HTML5 WebSocket*, 1st ed. Apress, 2013.

[31] R. Mallik, A. Hazarika, S. Ghosh, and D. Sing, ‘Development of an Android application for viewing Covid-19 containment zones and monitoring violators who are trespassing into it using Firebase and’, 2020. doi: 10.21203/rs.3.rs-28226/v1.

[32] J. Herzog, ‘Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman’, *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 51–52, Feb. 2015, doi: 10.1145/2693208.2693252.

[33] R. Malhotra and A. Chug, ‘Software Maintainability: Systematic Literature Review and

Current Trends', *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, no. 8. World Scientific Publishing Co. Pte Ltd, pp. 1221–1253, Oct. 01, 2016, doi: 10.1142/S0218194016500431.

[34] S. H, 'Performance testing: methodologies and tools', *J. Inf. Eng. Appl.*, vol. 1, 2011, Accessed: Jun. 27, 2020. [Online]. Available: https://www.academia.edu/download/13143430/11.Performance_Testing_Methodologies_and_Tools.pdf.

[35] G. McGraw, 'Software security', *IEEE Secur. Priv.*, vol. 2, no. 2, pp. 80–83, 2004, doi: 10.1109/MSECP.2004.1281254.

[36] G. Myers, *The Art of Software Testing*, 2nd ed., vol. 15, no. 2. 2004.

[37] N. Arílo, 'Introdução a teste de Software', *Eng. Softw. Mag.*, no. 1, 2007, Accessed: Jun. 27, 2020. [Online]. Available: http://www.comp.ita.br/~mluisa/TesteSw.pdf.

[38] P. B. Selvapriya, 'Different Software Testing Strategies and Techniques', *Int. J. Sci. Mod. Eng.*, vol. 2, no. 1, Dec. 2013, Accessed: Jun. 27, 2020. [Online]. Available: http://www.ijisme.org/wp-content/uploads/papers/v2i1/L05271111213.pdf.

[39] I. Jovanovic, 'Software Testing Methods and Techniques', *IPSI BgD Trans. Internet Res.*, vol. 5, no. 1, pp. 30–41, Jan. 2009, Accessed: Jun. 27, 2020. [Online]. Available: www.internetjournals.net.

[40] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, 'Software Testing Techniques: A Literature Review', in *6th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, Jan. 2016, pp. 177–182, doi: 10.1109/ict4m.2016.045.

[41] M. Parihar, 'Role of Testing in Software Development Life Cycle', *Int. J. Res.*, vol. 6, no. 8, 2019, doi: 10.26438/ijcse/v7i5.886889.

[42] L. Luo, 'Software testing techniques: Technology Maturation and Research Strategies', Pittsburgh, 2001. Accessed: Jul. 27, 2020. [Online]. Available: https://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf.

[43] Itti Hooda and Rajender Singh Chhillar, 'Software Test Process, Testing Types and Techniques', *Int. J. Comput. Appl.*, vol. 111, no. 13, pp. 975–8887, Feb. 2015, Accessed: Jul. 27, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.1299&rep=rep1&type=pdf.

[44] A. A. Sawant, P. H. Bari, and P. M. Chawan, 'Software Testing Techniques and Strategies', *Int. J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 980–986, May 2012, Accessed: Jul. 27, 2020. [Online]. Available: www.ijera.com.

[45] 'Automated Software Testing Tools for Windows and Linux | Software Automation'. .

[46] Syed Roohullah Jan, Syed Tauhid Ullah Shah, Zia Ullah Johar, Yasin Shah, and Fazlullah Khan, 'An Innovative Approach to Investigate Various Software Testing Techniques and Strategies', *Int. J. Sci. Res. Sci. Eng. Technol.*, vol. 2, no. 2, 2016, Accessed: Jul. 27, 2020. [Online]. Available: https://www.researchgate.net/publication/303280520.

[47] 'Introdução ao Teste de Software'. https://blog.nexxera.com/introducao-ao-teste-de-software/ (accessed Jul. 27, 2020).

[48] J. S. Carson, 'Model verification and validation', in *Proceedings of the Winter Simulation Conference*, Dec. 2002, vol. 1, pp. 52–58, doi:

10.1109/WSC.2002.1172868.

[49] H. K. N. Leung and L. White, ‘Insights into regression testing (software testing)’, in *Proceedings. Conference on Software Maintenance - 1989*, Oct. 1989, pp. 60–69, doi: 10.1109/icsm.1989.65194.

[50] ‘images (PNG Image, 439 × 115 pixels)’. https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcRE4MUjggsdkXoWwDxDkUoxI_qVT0rX6LaIZg&usqp=CAU (accessed Jul. 27, 2020).

[51] ‘Functional Testing Vs Non-Functional Testing’. https://www.softwaretestinghelp.com/functional-testing-vs-non-functional-testing/ (accessed Jun. 28, 2020).

[52] Renu Rajni and Pradeep Oak, *Software Testing: Effective Methods, Tools and Techniques*. McGraw Hill Education, 2004.

[53] I. Molyneaux, *The Art of Application Performance Testing*, 1st ed. O’Reilly Media, 2009.

[54] H. Kaur and G. Gupta, ‘Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and TestComplete’, *J. Eng. Res. Appl.*, vol. 3, no. 5, pp. 1739–1743, Sep. 2013, Accessed: Jul. 27, 2020. [Online]. Available: www.ijera.com.

[55] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat, ‘Classification of software testing tools based on the software testing methods’, in *2009 International Conference on Computer and Electrical Engineering, ICCEE 2009*, 2009, vol. 1, pp. 229–233, doi: 10.1109/ICCEE.2009.9.

[56] Muhammad Shahid and Suhaimi Ibrahim, ‘An Evaluation of Test Coverage Tools in Software Testing’, Feb. 2011, Accessed: Jul. 27, 2020. [Online]. Available: https://www.researchgate.net/publication/228922323_An_Evaluation_of_Test_Coverage_Tools_in_Software_Testing.

[57] PerfMatrix, ‘90th Percentile in Performance Testing | How to calculate 90th percentile |’, 2019. https://www.perfmatrix.com/90th-percentile-in-performance-testing/ (accessed Jul. 27, 2020).

[58] J. D. Summers, S. Joshi, and B. Morkos, ‘Requirements evolution: Relating functional and non-functional requirement change on student project success’, in *Proceedings of the ASME Design Engineering Technical Conference*, Jan. 2014, vol. 3, doi: 10.1115/DETC2014-35023.

[59] I. Sommerville, *Software engineering*, 10th ed. 2016.

[60] ‘C# Rises in Skills Report’s “Best-Known Language” Ranking -- Visual Studio Magazine’. https://visualstudiomagazine.com/articles/2020/02/05/hackerrank-2020.aspx (accessed Aug. 11, 2020).

[61] X. Decoster, ‘An Overview of the NuGet Ecosystem - CodeProject’, 2013. https://www.codeproject.com/Reference/628210/An-Overview-of-the-NuGet-Ecosystem (accessed Aug. 11, 2020).

[62] statianzo, ‘Fleck: C# Websocket Implementation’. Accessed: Aug. 11, 2020. [Online]. Available: https://github.com/statianzo/Fleck.

[63] ‘GitHub - jchristn/WatsonWebsocket: A simple C# async websocket server and client for reliable transmission and receipt of data’. https://github.com/jchristn/WatsonWebsocket (accessed Nov. 13, 2020).

[64] ‘GitHub - vtortola/WebSocketListener: A lightweight and scalable asynchronous

WebSocket listener'. https://github.com/vtortola/WebSocketListener (accessed Nov. 13, 2020).

[65] 'JavaScript | MDN'. https://developer.mozilla.org/en-US/docs/Web/JavaScript (accessed Oct. 19, 2020).

[66] 'CSS: Cascading Style Sheets | MDN'. https://developer.mozilla.org/en-US/docs/Web/CSS (accessed Oct. 19, 2020).

[67] 'html-css-javascript-905348.png (1160×498)'. https://www.internetingishard.com/html-and-css/introduction/html-css-javascript-905348.png (accessed Oct. 19, 2020).

[68] 'Apache JMeter - Apache JMeter™'. https://jmeter.apache.org/ (accessed Dec. 02, 2020).

[69] Naga Mallala, 'Neoload Tutorial: Neoload Introduction, Download And Installation', Nov. 13, 2020. https://www.softwaretestinghelp.com/neoload-tutorial/ (accessed Dec. 14, 2020).

[70] 'Load Testing Tool | NeoLoad by Neotys'. https://www.neotys.com/neoload/overview (accessed Dec. 02, 2020).

[71] A. Carlos De Francisco, 'Aquisição de competências no estágio curricular supervisionado: o caso dos cursos de engenharia do CEFET-PR', Florianópolis, SC, 2003. Accessed: Dec. 05, 2020. [Online]. Available: https://repositorio.ufsc.br/handle/123456789/85558.

[72] T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, vol. 3, no. 1. 2013.

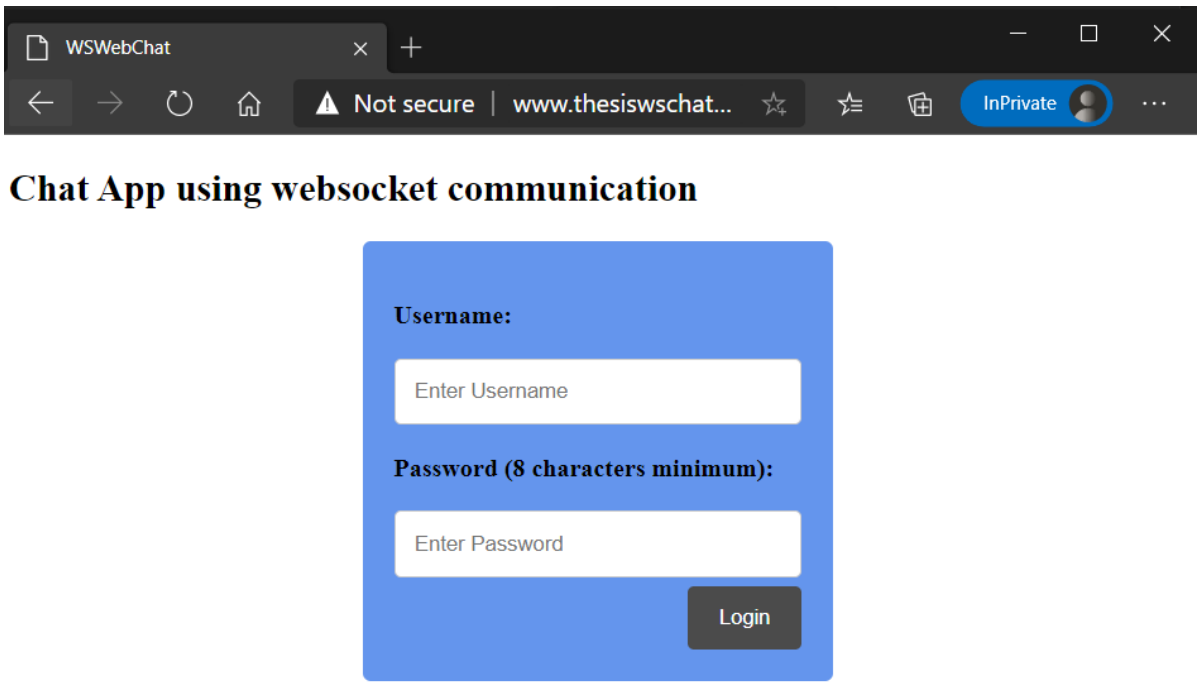
8. Anexos

8.1. Anexo A – Aplicação chat desenvolvida com comunicação WS

Desde a secção 8.1.1 até à 8.1.5 deste capítulo apresentar-se-ão todos os ecrãs desenvolvidos da aplicação de *chat*, bem como a utilização de todas as funcionalidades da mesma. No decurso das seguintes demonstrações. Passar-se-á pelo *login*, pelo momento em que o utilizador envia um anexo na sala de *chat* em que se encontra. Identificar-se-á notificações de novas mensagens e a criação de uma nova sala de *chat*. Por fim efetuar-se-á uma troca de mensagens entre utilizadores que permitirá apresentar a informação enviada pelos utilizadores dentro da área de *chat*.

8.1.1. Login

Para que o utilizador comece a disfrutar desta aplicação de *chat*, terá de fazer um *login* na aplicação, de modo que o sistema garanta que este tem autorização e caso assim o seja, o autenticar. Nesta versão aplicacional, não existe um ecrã de registo, pelo que, caso todos os utilizadores que não existirem no sistema com o nome de utilizador introduzido, serão criados e por sua vez passarão pelo processo de autorização e autenticação do sistema. Este *login* é realizado através do formulário da figura abaixo.

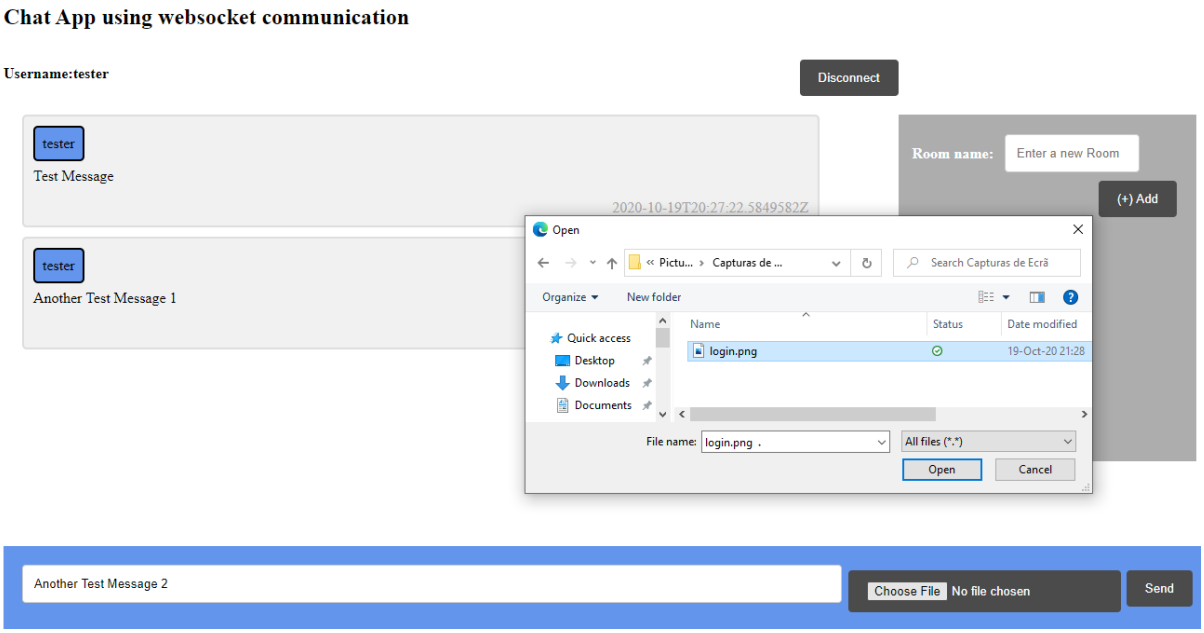


Estes dados são enviados para o servidor quando é estabelecida uma ligação de *WebSockets,* através do URL como parâmetros *de QueryString*. Sendo que foi pensado na utilização criptográfica para proteger a *password* através da utilização do algoritmo SHA-256 *Hash* de encriptação no campo *password* para que a palavra-passe do utilizador não fique exposta. Além disso, este mecanismo de segurança também é utilizado na vertente de base de dados, sendo que no momento da criação do utilizador é empregue, novamente, o mesmo mecanismo de encriptação com informação adicional. A razão desta encriptação deve-se ao facto de a *password* do utilizador ser gravada na base de dados, e como tentativa de salvaguardar a informação deste, em situações em que a base de dados seja exposta por pessoas mal intencionadas. Na vertente de servidor, essa encriptação é alcançada através do acréscimo de um campo adicional chamado de sal, que foi criado a partir do *username* e

um GUID (*Globally Unique Identifier*) gerado pelo servidor. Este sal é também encriptado em SHA256 e consequentemente é encriptada a password e gravada na base de dados com um menor risco de exposição de informação.

8.1.2. Envio de um anexo

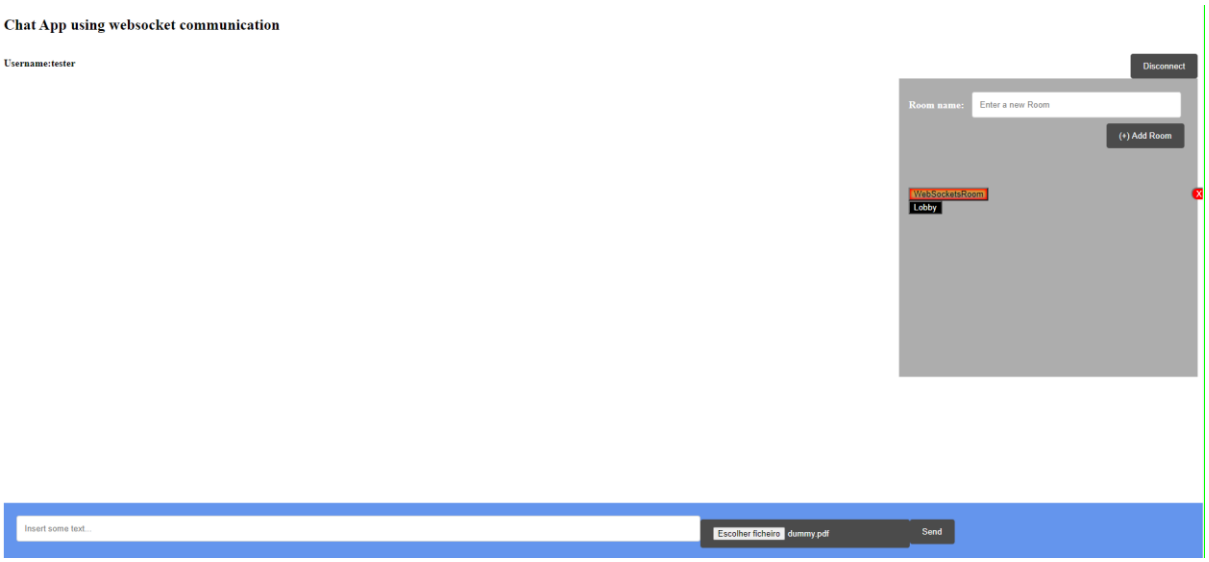
Ao autenticar-se com sucesso, o utilizador poderá trocar mensagens e anexos com outros utilizadores. A figura seguinte demonstra como poderá ser enviado um ficheiro para os utilizadores da mesma sala.



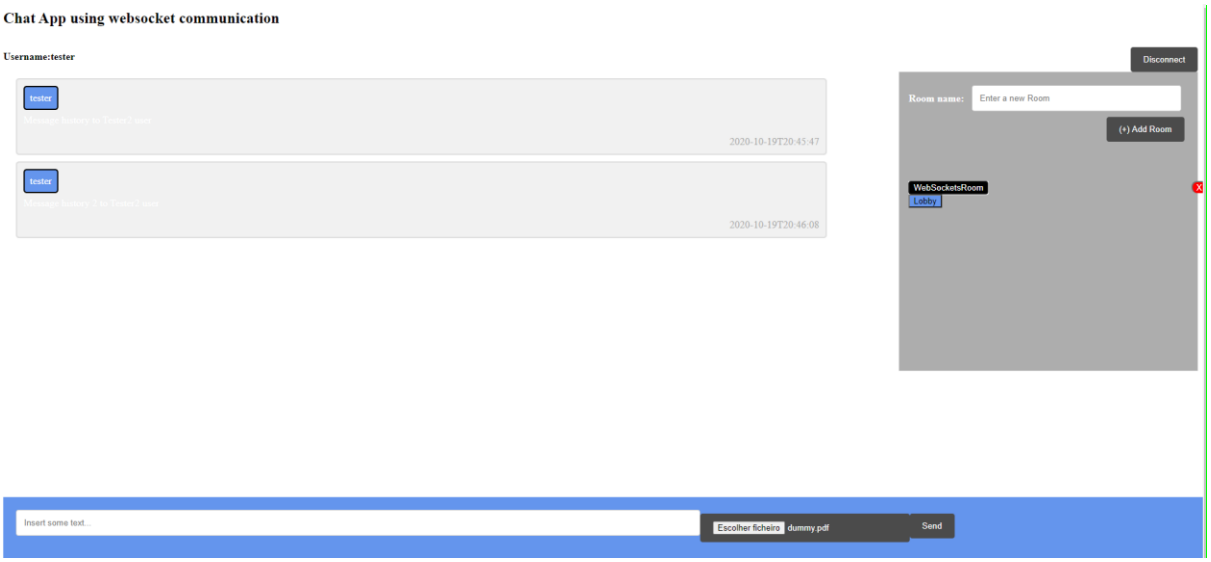
8.1.3. Alerta de novas notificações de mensagens

Em momentos que um utilizador se encontre numa outra sala de *chat* e receba novas mensagens de outra sala, a sala em que existem novas mensagens ficará com um realce em torno do botão correspondente, para que o utilizador compreenda que existe uma atualização para a sala de *chat*. A próxima figura demostra este realce em torno do botão da sala na qual foi criada na secção anterior. Podemos verificar na figura seguinte, que o utilizador poderá

sair de qualquer sala criada pelos utilizadores, sendo deste modo a sala *Lobby* a única sala comum a todos.



Ao clicar em cima do botão da sala que possui notificações, o utilizador alterará para a sala de *chat* onde poderá verificar as novas mensagens e, por fim, remover, automaticamente, o realce em torno do botão. A figura seguinte representa as notificações das novas mensagens que o utilizador possuía por ler.



8.1.4. Criação de uma nova sala de *chat*

Sempre que o utilizador assim o pretender, poderá criar uma sala de *chat*, ou então aceder a uma já exista. Na figura que se segue, na área lateral direita a cinzento, existe um

campo de entrada de texto que permite ao utilizador introduzir o nome da sala que pretende criar ou aceder e em seguida terá de carregar no botão azul com nome *Add*.

Assim que o utilizador premir o botão *Add*, a aplicação adicionará um novo botão correspondente à sala adicionada, tal como podemos verificar na figura seguinte.

8.1.5. Troca de mensagens entre utilizadores

As trocas de mensagens entre utilizadores nas salas da aplicação são caraterizadas por três parâmetros: o nome, o conteúdo da mensagem escrita e a data. Além disso, para diferenciar as mensagens do utilizador autenticado dos restantes presentes na sala, é utilizado um fundo cinza claro. Além disso, outra componente que permite identificar o autor

da mensagem é o nome. A figura seguinte serve como complemento ao que foi acima descrito.