UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

HERIK LIMA DE CASTRO

Um *web framework* para C++: projeto, implementação e avaliação

São Paulo

2019

HERIK LIMA DE CASTRO

Um *web framework* para C++: projeto, implementação e avaliação

Dissertação apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação.

Área de concentração: Engenharia de Software

Versão corrigida contendo as alterações solicitadas pela comissão julgadora em 11 de outubro de 2019. A versão original encontra-se em acervo reservado na Biblioteca da EACH-USP e na Biblioteca Digital de Teses e Dissertações da USP (BDTD), de acordo com a Resolução CoPGr 6018, de 13 de outubro de 2011.

Orientador: Prof. Dr. Marcelo Medeiros Eler

São Paulo

2019

Autorizo a reprodução e divulgação total ou parcial deste trabalho, por qualquer meio convencional ou eletrônico, para fins de estudo e pesquisa, desde que citada a fonte.

CATALOGAÇÃO-NA-PUBLICAÇÃO

(Universidade de São Paulo. Escola de Artes, Ciências e Humanidades. Biblioteca)
CRB 8 - 4936

Castro, Herik Lima de

Um *web framework* para C++: projeto, implementação e avaliação / Herik Lima de Castro ; orientador, Marcelo Medeiros Eler. – 2019.

101 f. : il

Dissertação (Mestrado em Ciências) - Programa de Pós-Graduação em Sistemas de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo.

Versão corrigida

1. Engenharia de software. 2. Desenvolvimento de software. 3. Engenharia da web. 4. C++ (linguagem de programação). 5. Frameworks. I. Eler, Marcelo Medeiros, orient. II. Título

CDD 22.ed.– 005.1

Dissertação de autoria de Herik Lima de Castro, sob o título “Um *web framework* para C++: projeto, implementação e avaliação”, apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação, na área de concentração Engenharia de Software, aprovada em 11 de outubro de 2019 pela comissão julgadora constituída pelos doutores:

Prof. Dr. Marcelo Medeiros Eler

Instituição: Universidade de São Paulo

Presidente

Prof. Dr. Daniel de Angelis Cordeiro

Instituição: Universidade de São Paulo

Prof. Dr. Calebe de Paula Bianchini

Instituição: Universidade Presbiteriana Mackenzie

Prof. Dr. Emilio de Camargo Franceschini

Instituição: Universidade Federal do ABC

Eu dedico essa dissertação de mestrado à pessoa mais especial que já encontrei na minha vida, a minha falecida noiva, Camila Angélica Calazans de Sá, que me deixou em 08 de Junho de 2018, um mês antes de nosso casamento. Você foi a pessoa mais incrível e corajosa que conheci e sem você eu não teria chegado até aqui e por isso sou muito grato aos quase 6 anos que passamos juntos. Nesse tempo, você foi minha aluna e também minha professora. Você foi minha noiva, amante e melhor amiga. Nos tempos difíceis, você sempre esteve para me ajudar levantar. Esses anos que passei com você foram os melhores anos da minha vida. Lamento muito não tê-la mais ao meu lado, minha princesinha, mas saiba que, embora tenha partido, você jamais será esquecida e sempre viverá em meu coração.

Te amarei para sempre, minha princesinha!

Dedicatória

Agradecimentos

Agradeço aos professores Daniel de Angelis Cordeiro, Fátima de Lourdes dos Santos Nunes Marques, Karina Valdivia Delgado, Luciano Antonio Digiampietri, Luciano Vieira de Araújo, Patrícia Rufino Oliveira e Patrícia Rufino Oliveira pelas excelentes disciplinas ministradas no PPgSI. Essas disciplinas foram fundamentais para a elaboração dessa dissertação. Aproveito também para agradecer ao Daniel de Angelis Cordeiro e ao João Bernardes por terem participado da banca de qualificação; pois eles trouxeram inúmeras críticas construtivas que ajudaram a melhorar, e muito, o trabalho final. Também sou muito grato ao meu orientador, Marcelo Medeiros Eler, que sempre foi um orientador muito presente, atencioso e competente. Sem ele, esse trabalho não teria chegado até aqui. Aproveito também para agradecer minha mãe, Susana de Jesus Lima, e meu irmão, Adrian Lima de Castro, pelo apoio que me deram após a perda da minha noiva e também à AGIT Informática Ltda, em especial aos meus chefes, Basílio Miranda e Jussara Hugges, que cederam horários para que eu pudesse cursar as disciplinas e demais atividades relacionadas ao PPgSI. A todos vocês que me ajudaram a chegar até aqui, meu muitíssimo obrigado!

“O impossível existe até que alguém duvide dele e prove o contrário.”

(Albert Einstein)

Resumo

CASTRO, Herik Lima. **Um web framework para C++**: projeto, implementação e avaliação. 2019. 101 f. Dissertação (Mestrado em Ciências) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2019.

As aplicações *web* são, tipicamente, programas do tipo cliente-servidor em que os clientes consomem serviços fornecidos por um servidor *web*. As aplicações *web* podem ser tão simples quanto sites que exibem conteúdos estáticos e dinâmicos, quanto aplicações que envolvem o uso de banco de dados, interação com outros sistemas de informação e o processamento de algoritmos complexos e grandes volumes de dados. Em geral, para desenvolver aplicações mais complexas, até mesmo desenvolvedores mais experientes enfrentam vários desafios: *configuração complexa e difícil aprendizagem*, pois precisam utilizar diferentes linguagens de programação, bibliotecas, *frameworks*, arquivos de configuração e código específico para tornar múltiplas camadas interoperáveis; *baixa manutenibilidade*, em razão da complexa configuração, conflitos entre bibliotecas e *frameworks* e do entrelaçamento entre código de negócio, de controle e de apresentação permitido em diversas soluções; *alto consumo de recursos computacionais*, em razão do custo de processamento e comunicação introduzido pelo grande número de componentes de uma arquitetura *web*; e *baixo desempenho*, em razão da alta dependência de linguagens interpretadas e da comunicação entre diversos módulos, bibliotecas e *frameworks*. Neste contexto, o objetivo principal deste projeto de pesquisa é propor e avaliar uma solução para o desenvolvimento *web* que mitigue os principais problemas relacionados ao cenário atual e mencionados anteriormente. A solução foi apresentada na forma de um novo *web framework* chamado *CWF* (*C++ Web Framework*), que apoia o desenvolvimento de aplicações escritas na linguagem C++. As decisões de projeto do *web framework* proposto tiveram como objetivo alcançar alto desempenho por utilizar uma linguagem compilada; baixo consumo de recursos computacionais (memória e processamento); mecanismos simples de configuração; separação da camada de apresentação (*HTML*, por exemplo) da camada de negócios (código *back-end*); e ser multiplataforma. O *web framework* foi avaliado quanto à sua facilidade de instalação, configuração e uso por meio de experimentos com usuários, e também foram realizados testes de desempenho e consumo de recursos computacionais. Os resultados dão indícios de que o *CWF* é fácil de instalar, configurar, usar e apresenta bom desempenho no que diz respeito ao tempo de resposta e consumo de recursos como memória *RAM* e processador. Além disso, ele já tem sido utilizado no desenvolvimento de aplicações reais, tanto no Brasil, quanto no exterior.

Palavras-chave: *web framework*, bibliotecas para desenvolvimento *web*, avaliação, desempenho, manutenibilidade, consumo de memória, tempo de resposta, *C++ Web Framework*.

Abstract

CASTRO, Herik Lima. **A C++ web framework:** project, implementation and evaluation. 2019. 101 p. Dissertation (Master of Science) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2019.

Web applications are, typically, client-server programs where clients consume services provided by a web server. Web applications can be as simple as portals that display static and dynamic content as applications involving database use, interaction with other information systems, and the processing of complex algorithms and large data volumes. In general, to develop more complex applications, even more experienced developers face a number of challenges: *complex configuration and difficult to learn* as they need to use different programming languages, libraries, frameworks, configuration files and specific code to make multiple layers interoperable; *low maintainability*, due to the complex configuration, conflicts between libraries and *frameworks* and the intertwining of business, control and presentation code allowed in various solutions; *high computational resource consumption* due to processing and communication cost introduced by the large number of components of a webtext architecture; and *poor performance*, due to the high dependence on interpreted languages and the communication between several modules, libraries and frameworks. In this context, the main objective of this research project is to propose and evaluate a solution for web development that mitigates the main issues related to the current scenario mentioned above. The solution was presented in the form of a new web framework called CWF, which supports the development of applications written in the C++ language. The design decisions of the proposed web framework were to achieve high performance by using a compiled language; low consumption of computational resources (memory and processing); simple configuration mechanisms; separation of the presentation layer (HTML, for example) from the business layer (backend code); and being cross-platform. The web framework was evaluated for its ease of installation, configuration and use through user experiments, as well as performance tests and computational resource consumption. The results give evidence that CWF is easy to install, configure, use and performs well with response time and resource consumption such as RAM and processor. In addition, it has already been used in the development of real applications, both in Brazil and abroad.

Keywords: *web framework, libraries for web development, evaluation, performance, maintenance, memory consumption, response time, C++ Web Framework.*

Lista de figuras

Figura 1 – Funcionamento de uma aplicação <i>web</i>	22
Figura 2 – Arquitetura do <i>CWF</i>	43
Figura 3 – Diagrama de Classes	44
Figura 4 – A estrutura de pastas de um projeto <i>CWF</i>	45
Figura 5 – Resultado da requisição	47
Figura 6 – Escolaridade dos participantes	60
Figura 7 – Experiência no setor de T.I.	60
Figura 8 – Experiência com desenvolvimento web	61
Figura 9 – Nível de dificuldade para se configurar e instalar o <i>CWF</i>	61
Figura 10 – Escolaridade dos participantes	66
Figura 11 – Experiência no setor de T.I (Desenvolvimento de Sistemas)	66
Figura 12 – Experiência com desenvolvimento <i>web</i>	67
Figura 13 – Experiência com desenvolvimento em C++	67
Figura 14 – Nível de dificuldade para se configurar e instalar o <i>CWF</i>	68
Figura 15 – Tempo gasto para resolver o primeiro exercício	68
Figura 16 – Tempo gasto para resolver o segundo exercício	69
Figura 17 – Tempo gasto para resolver o terceiro exercício	69
Figura 18 – Tempo gasto para resolver o quarto exercício	70
Figura 19 – Escolaridade dos participantes	74
Figura 20 – Experiência no setor de T.I	74
Figura 21 – Experiência com desenvolvimento web	75
Figura 22 – Experiência com desenvolvimento em C++	75
Figura 23 – Nível de dificuldade para se configurar e instalar o <i>CWF</i>	76
Figura 24 – Tempo gasto para resolver o primeiro exercício	76
Figura 25 – Tempo gasto para resolver o segundo exercício	76
Figura 26 – Tempo gasto para resolver o terceiro exercício	77
Figura 27 – Tempo gasto para resolver o quarto exercício	77
Figura 28 – Conseguiu solucionar o exercício com <i>GET</i>	78
Figura 29 – Conseguiu solucionar o exercício com <i>POST</i>	78
Figura 30 – Conseguiu solucionar o exercício com <i>PUT</i>	78

Figura 31 – Conseguiu solucionar o exercício com <i>DELETE</i>	79
Figura 32 – Fatorial (10) - Média	82
Figura 33 – Primo (997) - Média	83
Figura 34 – Fibonacci (20) - Média	84
Figura 35 – Pages (10000) - Média	85
Figura 36 – Funcionamento da aplicação	90
Figura 37 – Número de acessos no <i>GitHub</i>	93

Lista de algoritmos

Algoritmo 1 – Exemplo de uma aplicação <i>CGI</i> em C++	23
Algoritmo 2 – Exemplo de um <i>template</i> em <i>ColdFusion</i> para listar os nomes dos clientes de uma tabela no banco de dados	24
Algoritmo 3 – Exemplo de um <i>Hello World</i> em <i>Velocity</i>	25
Algoritmo 4 – Exemplo de uma página em PHP	25
Algoritmo 5 – Exemplo de uma aplicação <i>web</i> em <i>Java EE</i>	28
Algoritmo 6 – Iniciando o servidor <i>web Wt</i>	29
Algoritmo 7 – Criando uma <i>WApplication</i> uma aplicação em <i>Wt: Web Tool Kit</i>	29
Algoritmo 8 – Exemplo de uma aplicação em <i>QtWebApp</i>	30
Algoritmo 9 – Exemplo de uma página em <i>QtWebApp</i>	31
Algoritmo 10 – Exemplo de uma página <i>web</i> utilizando <i>POCO C++ Libraries</i>	31
Algoritmo 11 – Exemplo de uma aplicação <i>web</i> em <i>Crow</i>	32
Algoritmo 12 – Arquivo de configuração do <i>CWF</i> (CPPWeb.ini)	46
Algoritmo 13 – <i>View helloworld</i>	46
Algoritmo 14 – Criando uma aplicação <i>web</i> em <i>CWF</i>	47
Algoritmo 15 – Exemplo de uso de controllers e sessões	49
Algoritmo 16 – Exemplo de um Filtro em <i>CWF</i>	50
Algoritmo 17 – Uma classe <i>Model</i> preparada para ser utilizada pela <i>CSTL</i>	51
Algoritmo 18 – Enviando objetos para a página via <i>Controller</i>	52
Algoritmo 19 – Uso da <i>CSTL for, if</i> e <i>out</i> na <i>View</i>	52
Algoritmo 20 – Exemplo de uma classe <i>model</i> para ser utilizada no módulo <i>ORM</i>	53
Algoritmo 21 – Exemplo de uso do módulo <i>ORM</i>	54

Lista de tabelas

Tabela 1 – Distribuição do tempo gasto em cada exercício por nível de experiência com desenvolvimento <i>web</i>	71
Tabela 2 – Fatorial (10) - Resultados	81
Tabela 3 – Primo (997) - Resultados	82
Tabela 4 – Fibonacci (20) - Resultados	83
Tabela 5 – <i>Pages</i> (10000) - Resultados	84

Lista de abreviaturas e siglas

ASP	Active Server Pages
CGI	Common Gateway Interface
CPD	Centro de Processamento de Dados
CSP	C++ Server Pages
CSTL	C++ Server Pages Standard Tags Library
CWF	C++ Web Framework
ERP	Enterprise Resource Planning
HTML	Hypertext Markup Language
HTTP	Hipertext Transfer Protocol
HTTPS	Hipertext Transfer Protocol Secure
IDE	Integrated Development Environment
IP	Internet Protocol
ISO	International Organization for Standardization
Java EE	Java Enterprise Edition
JSON	JavaScript Object Notation
JSP	Java Server Pages
JSTL	Java Server Pages Standard Tags Library
JVM	Java Virtual Machine
MIT	Massachusetts Institute of Technology
moc	MetaObject Compiler
MVC	Model-View-Controller
ORM	Object Relational Mapper

POCO	Portable Components
QML	Qt Modeling Language
RAM	Random Access Memory
RTOS	Real-Time Operating Systems
REST	Representational State Transfer
SaaS	Software as a Service
SQL	Structured Query Language
SSI	Server-Side Includes
SSL	Secure Sockets Layer
TCP	Transport Layer Protocol
URL	Uniform Resource Locator
VB	Visual Basic
XHTML	eXtensible Hypertext Markup Language
XML	eXtensible Markup Language

Sumário

1	Introdução	17
1.1	<i>Motivação</i>	17
1.2	<i>Objetivos</i>	19
1.3	<i>Metodologia</i>	20
1.4	<i>Organização</i>	21
2	Referencial teórico	22
2.1	<i>Aplicações web</i>	22
2.2	<i>Web frameworks</i>	26
2.2.1	<i>Java Enterprise Edition</i>	27
2.2.2	<i>Wt: Web Tool Kit</i>	28
2.2.3	<i>QtWebApp</i>	30
2.2.4	<i>POCO C++ Libraries</i>	31
2.2.5	<i>Crow</i>	32
2.3	<i>Análise de desempenho</i>	32
2.4	<i>Considerações finais</i>	34
3	CWF - C++ Web Framework	35
3.1	<i>Requisitos mínimos</i>	35
3.2	<i>Decisões de projeto</i>	36
3.2.1	<i>Linguagem de programação</i>	36
3.2.2	<i>Framework de apoio</i>	37
3.2.3	<i>Arquivos de configuração</i>	40
3.2.4	<i>Separação entre código de apresentação e negócios</i>	40
3.2.5	<i>Contêiner de aplicação</i>	41
3.2.6	<i>Inspiração no Java Servlets</i>	41
3.3	<i>Descrição do C++ Web Framework (CWF)</i>	42
3.3.1	<i>A arquitetura do CWF</i>	43
3.3.2	<i>A estrutura de um projeto CWF</i>	44
3.3.3	<i>As classes CppWebApplication e CppWebServer</i>	45
3.3.4	<i>Controllers</i>	48

3.3.5	Páginas web e a CSTL	51
3.4	<i>Acesso a Banco de Dados</i>	53
3.5	<i>Considerações finais</i>	55
4	Avaliação	56
4.1	<i>Avaliação da instalação e configuração inicial</i>	56
4.2	<i>Avaliação da facilidade de aprendizagem e uso</i>	59
4.2.1	Primeira etapa: instalação e criação de página estática	59
4.2.2	Segunda etapa: instalação, configuração e criação de página dinâmica	62
4.2.3	Terceira etapa: criação de serviços web e banco de dados	72
4.3	<i>Avaliação de desempenho</i>	79
4.4	<i>A avaliação de viabilidade e aplicabilidade</i>	86
4.4.1	Questionários aplicados a usuários do GitHub	86
4.4.2	Uso em ambiente de produção real	89
4.5	<i>Página web e divulgação</i>	92
4.6	<i>Considerações finais</i>	93
5	Conclusões	94
5.1	<i>Resultados</i>	95
5.2	<i>Contribuições</i>	96
5.3	<i>Trabalhos futuros</i>	97
	Referências¹	98

¹ De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

1 Introdução

As aplicações *web* são, tipicamente, programas do tipo cliente-servidor em que os clientes, geralmente um navegador, consomem serviços fornecidos por um servidor *web*. Aplicações *web* podem ser tão simples quanto sites que exibem conteúdos estáticos e dinâmicos, quanto sistemas de informação complexos que envolvem o uso de banco de dados, interação com outros sistemas de informação e o processamento de algoritmos complexos e grande volumes de dados. Desde seu surgimento, as aplicações *web* tornaram-se mais robustas e foram adotadas como uma das principais plataformas de operações de muitas organizações públicas e privadas, tornando-se um grande marco na evolução tecnológica de nossa história (CHAUBEY; SURESH, 2001; GINIGE; MURUGESAN, 2001; OFFUTT, 2002).

O aumento da complexidade e da demanda por aplicações *web* cresceu com a popularização de tecnologias e soluções relacionadas a *Software* como Serviço (SaaS) (TSAI; BAI; HUANG, 2014), Computação em Nuvem (BUYIA et al., 2009) e Aplicações Móveis (CHARLAND; LEROUX., 2011). Naturalmente, as linguagens de programação, as bibliotecas e os *web frameworks* para o desenvolvimento de aplicações *web* também evoluíram para acompanhar as demandas tecnológicas e de negócios.

1.1 Motivação

O aumento da complexidade das aplicações *web* e dos recursos de apoio ao seu desenvolvimento trouxeram consigo vários impactos para o contexto de desenvolvimento deste tipo de aplicação: configuração complexa, difícil aprendizagem, baixa manutenibilidade, baixo desempenho e alto consumo de recursos computacionais. Esses problemas são detalhados a seguir.

Configuração complexa e difícil aprendizagem. Escrever até mesmo aplicações *web* simples requer um grande esforço, pois, em geral, desenvolvedores *web* precisam utilizar diferentes linguagens de programação, bibliotecas e *frameworks* em um mesmo projeto, e também lidar com conflitos entre eles. Além disso, o desenvolvimento de uma aplicação *web* pode envolver diversos arquivos de configuração e código específico para tornar as múltiplas camadas interoperáveis (SINHA; KARIM; GUPTA, 2015; SALAS-ZARATE et

al., 2015; VUORIMAA et al., 2016). Consequentemente, a dificuldade para se desenvolver aplicações *web* é alta mesmo para desenvolvedores experientes (SINHA; KARIM; GUPTA, 2015; SWAIN et al., 2016).

Baixa manutenibilidade. O uso de múltiplas bibliotecas, *frameworks* e arquivos de configuração também pode dificultar a manutenibilidade das aplicações *web*, pois localizar e corrigir defeitos, criar novas funções, integrar soluções e manter bibliotecas atualizadas e compatíveis entre si pode ser um grande desafio (RAEMAEEKERS; DEURSEN; VISSER, 2014). Por outro lado, as alternativas mais simples para o desenvolvimento *web*, em geral, consistem em criar arquivos independentes em um servidor *web*, que é capaz ler, processar e responder requisições de clientes utilizando estes arquivos. Uma das limitações desse tipo de abordagem, entretanto, é o fato de que ela pode gerar problemas de manutenibilidade por misturar constantemente a camada de negócios com a camada de apresentação (SHKLAR; ROSEN, 2004), entrelaçando o código *HTML* com o código da linguagem de programação.

Alto consumo de recursos computacionais.

O consumo de recursos computacionais como memória e processador dos *frameworks* e aplicações *web* atuais pode não ser satisfatório em razão do custo de processamento e comunicação introduzido pelo grande número de componentes de uma arquitetura *web*. Embora os recursos computacionais estejam ficando cada vez mais baratos, principalmente com o uso de computação em nuvem, há situações em que a economia de recursos pode ser desejável, como, por exemplo, em razão da economia de energia elétrica altamente associada ao uso de processador e memória.

Baixo desempenho. A dinamicidade das aplicações *web* geralmente requer o uso de linguagens de programação interpretadas, pois o processo de alterar o comportamento da aplicação exige menos esforço do que fazer o mesmo com linguagens de programação compiladas. Em linguagens de programação compiladas, pode ser necessário ter que encerrar a aplicação, alterar seu código fonte, recompilar e colocar a aplicação novamente no ar. As linguagens de programação compiladas fornecem mecanismos para alterar as aplicações dinamicamente por meio de *plugins*, objetos ou componentes remotos e bibliotecas dinâmicas. Entretanto, este processo é mais trabalhoso, exigindo mais etapas do que simplesmente editar e salvar um arquivo. Por outro lado, as linguagens de programação interpretadas podem ser até 84 vezes mais lentas do que linguagens de programação compiladas (LUBIN; DUNNING, 2015).

Embora o desempenho das aplicações não seja a maior preocupação no desenvolvimento *web*, existem aplicações ou funcionalidades específicas em que um bom desempenho é altamente desejável.

Curiosamente, embora as linguagens de programação C e C++ sejam conhecidas pelo seu bom desempenho e baixo consumo de recursos computacionais em comparação a outras linguagens de programação (GAME, 2018), e também estejam no top 4 das linguagens de programação mais utilizadas no mundo (TIOBE, 2019), respectivamente, elas não aparecem nem entre as cinco principais linguagens de programação mais utilizadas no desenvolvimento *web* (MILLARES, 2015).

Uma das razões para a baixa adoção das linguagens de programação C e C++ em ambientes de desenvolvimento *web*, além da forte concorrência de soluções mais maduras e estabelecidas na indústria, são as limitações dos *web frameworks* e bibliotecas *web* para desenvolvimento C e C++ existentes. Uma análise dos mais conhecidos *web frameworks* e bibliotecas para desenvolvimento *web* em C++, como *POCO Libraries*, *Wt*, *Crow* e *QWebApp*, por exemplo, permite constatar que nenhum deles faz uma verdadeira separação da lógica de negócios da camada de apresentação, prejudicando, assim, a adaptabilidade, manutenibilidade e reusabilidade (SRAI et al., 2017).

1.2 Objetivos

O objetivo principal deste projeto de pesquisa é propor e avaliar uma solução para o desenvolvimento *web* que mitigue os principais problemas relacionados ao cenário atual, a saber: configuração complexa, difícil aprendizagem, baixa manutenibilidade, alto consumo de recursos computacionais e baixo desempenho. Os objetivos específicos são os seguintes:

- i) realizar o levantamento dos principais problemas relacionados ao desenvolvimento *web*;
- ii) desenvolver um protótipo de um *web framework* que mitigue os problemas levantados;
- iii) e avaliar a solução proposta.

1.3 Metodologia

As atividades necessárias para alcançar os objetivos deste trabalho são apresentadas, a seguir, de forma resumida. Em primeiro lugar, foi realizada uma revisão de literatura e estudo dos principais *web frameworks* disponíveis atualmente para entender os principais conceitos, características e limitações relatadas por pesquisadores e profissionais da área; e identificar os recursos mínimos necessários que devem ser oferecidos por um *web framework* para o desenvolvimento de uma aplicação *web*. Em seguida, foram definidos os requisitos e a arquitetura do protótipo do *web framework* proposto neste projeto de pesquisa, de tal forma que os problemas encontrados na literatura fossem mitigados ou solucionados. A atividade seguinte foi a implementação do protótipo do *web framework* com base nos requisitos e nas arquiteturas definidas. A última etapa do projeto, por sua vez, envolveu a avaliação de diferentes aspectos do *web framework* proposto:

- A avaliação da complexidade de configuração foi feita por meio da comparação da quantidade de arquivos e bibliotecas necessárias para desenvolver aplicações em diversos *web frameworks* disponíveis atualmente;
- A avaliação da facilidade de aprendizagem e uso foi realizada com 37 desenvolvedores com diferentes níveis de experiência em desenvolvimento e desenvolvimento *web*;
- A avaliação do desempenho e do consumo de recursos computacionais foi realizada em diversas situações, como carregamento de páginas *web* dinâmicas e execução de algoritmos complexos com acesso a banco de dados, por exemplo. Nesta avaliação, foi feito o teste de carga, a medida do consumo de processador e memória *RAM* do servidor e o tempo de resposta;
- A avaliação de sua viabilidade e aplicabilidade em ambiente real foi feita de duas maneiras. Primeiramente, por meio da disponibilização do protótipo no *GitHub* e posterior aplicação de questionário para usuários do protótipo; e implementação de aplicação real utilizando o *web framework* proposto. A avaliação da manutenibilidade das aplicações reais não foi realizada em razão da limitação do tempo desta pesquisa.

1.4 Organização

Esta dissertação de mestrado está organizada da seguinte forma. No Capítulo 2, são apresentados os principais conceitos relacionados ao desenvolvimento de uma aplicação *web*. Neste capítulo, são discutidos os principais tipos de aplicação *web*, as principais arquiteturas e os principais *frameworks* de desenvolvimento *web* existentes no mercado. Também são discutidos os principais problemas encontrados nas tecnologias para desenvolvimento *web* atuais. No Capítulo 3, é apresentado o *web framework* proposto neste projeto de pesquisa, o *C++ Web Framework (CWF)*. O Capítulo 4 descreve o planejamento e execução das avaliações utilizando o *web framework* proposto. Por fim, as considerações finais do trabalho são apresentadas no Capítulo 5.

2 Referencial teórico

Neste capítulo, são apresentados os fundamentos teóricos para o entendimento desta pesquisa. Em um primeiro momento, disserta-se sobre aplicações *web* e seu funcionamento. Em seguida, disserta-se sobre as principais formas utilizadas para o desenvolvimento *web*. Por fim, são apresentados alguns exemplos de *web frameworks*.

2.1 Aplicações web

As aplicações *web* são, tipicamente, aplicações cliente-servidor em que o cliente, normalmente, utiliza o protocolo *HTTP* para enviar requisições e receber respostas de um servidor *web* (SHKLAR; ROSEN, 2004). As aplicações *web* podem ser acessadas pelos usuários por meio de um navegador, ou por aplicações que podem consumir serviços e recursos de uma aplicação *web*, como, por exemplo, *web services* (PAUTASSO, 2013). O *HTTP* é um protocolo de transferência de textos que atua sobre o *TCP/IP* para transmitir mensagens entre o cliente e o servidor (ZHANG et al., 2017). Essas mensagens contêm informações a respeito da requisição, como, por exemplo, a *URL* desejada, parâmetros e *cookies*. Um servidor *web* consiste em uma máquina com um servidor *HTTP* instalado, como *Nginx* (FJORDVALD; NEDELICU, 2018), *Apache* (BOWEN; COAR, 2018), *ISS* (KHAN, 2017) ou similares, que, ao receber uma requisição, processa a mensagem *HTTP* recebida para enviar uma resposta ao cliente. Essa resposta poderá ser uma página *HTML*, um arquivo *CSS* ou *Javascript*, uma imagem, um documento, etc; como mostra a Figura 1.

Figura 1 – Funcionamento de uma aplicação *web*

Fonte: <https://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888>

Embora as aplicações *web* sejam tipicamente cliente/servidor, e a essência de seu funcionamento seja o mesmo, elas podem ser construídas com diferentes ferramentas e abordagens. Atualmente, existem quatro abordagens que podem ser utilizadas no desenvolvimento *web*: *Scripting*, *Templates*, Híbridas e *web frameworks* (SHKLAR; ROSEN, 2004). As três primeiras abordagens são brevemente descritas a seguir. A abordagem de desenvolvimento *web* baseada em, *web frameworks*, está descrita em detalhes na próxima seção.

Em uma abordagem de *scripting* ou programática, o código associado a uma página consiste, predominantemente, de código escrito em Perl, Python, Tcl, C/C++ ou uma linguagem de programação de alto nível como *Java*. Enquadram-se nessa categoria: *CGI*, *FastCGI* e *Java Servlets*. Embora essa abordagem ofereça um meio interessante de se gerar páginas *web* dinâmicas, ela gera problemas na manutenibilidade das aplicações uma vez que necessita de uma constante mistura entre a camada de negócios com a camada de apresentação, algo que gera vários problemas para manter e evoluir a aplicação *web* (SHKLAR; ROSEN, 2004).

O Algoritmo 1 apresenta uma pequena aplicação feita em C++, que pode ser compilada e inserida dentro de um servidor com suporte a *CGI*. Após ser inserida no servidor e receber uma chamada, a aplicação será executada e a sua saída padrão, *cout*, será redirecionada para o servidor que, em seguida, irá redirecionar a página em *HTML* para o cliente que fez a requisição.

Algoritmo 1 Exemplo de uma aplicação *CGI* em C++

```

1: #include <iostream>
2: int main()
3: {
4:     std::cout << "Content - type : text/html\r\n\r\n";
5:     std::cout << "< html >< body > HelloWorld! < /body >< /html >";
6:     return 0;
7: }
```

Esta abordagem nos traz dois problemas: em primeiro lugar, ela induz o Desenvolvedor a misturar as camadas de apresentação (código *HTML*) com a camada de negócios (código C++), pois, sem isso, a tarefa de gerar páginas dinâmicas se tornará praticamente impossível. Em segundo lugar, a cada nova requisição a essa aplicação, o servidor *web* criará inúmeras variáveis de ambiente, e irá configurá-las com os parâmetros da requisição *HTTP* do cliente e iniciar uma nova aplicação. Isso causa uma perda de desempenho no

questo de tempo de resposta da aplicação, além de fazer com que o consumo de memória *RAM* e processamento aumentem significativamente, uma vez que, para cada requisição, o servidor irá configurar um conjunto de variáveis de ambiente e iniciar uma nova aplicação. Os *Java Servlets* resolveram o problema de desempenho que havia com o *CGI*, pois ele, ao invés de iniciar uma aplicação inteira, cria um novo *thread* (tarefa) para cada nova requisição, mas, sozinho, não consegue acabar com a dependência de abordagens centradas em código para a geração de páginas dinâmicas.

A abordagem de *template* utiliza um objeto (o *template*) que consiste predominantemente de estruturas formatadas, com construções embutidas e limitadas que adicionam poder programático. Nessa categoria, existem ferramentas como: *Server-Side Includes (SSI)*, *ColdFusion* e *Velocity*. Essa abordagem separa a camada de negócios da camada de apresentação, mas não tem o mesmo poder programático e flexibilidade das abordagens híbridas (SHKLAR; ROSEN, 2004).

O Algoritmo 2 apresenta um *template* criado em *ColdFusion*. Esse *template* irá selecionar os nomes de todos os clientes, presentes na tabela clientes, e, em seguida, montar uma tabela em *HTML*, contendo os resultados para serem apresentados a quem fez a requisição.

Algoritmo 2 Exemplo de um *template* em *ColdFusion* para listar os nomes dos clientes de uma tabela no banco de dados

```

1: < CFQUERY DATASOURCE = "oracle - prod" NAME = "dbquery1" >
2:   SELECT NOME FROM CLIENTES;
3: < /CFQUERY >
4: ...
5: < TABLE >
6:   < TR >
7:     < TD ALIGN = "center" > < B > Name < /B > < /TD >
8:   < /TR >
9:   < TR >
10:    < CFOUTPUT QUERY = "dbquery1" >
11:      < TD > #NOME# < /TD >
12:    < /CFOUTPUT >
13:   < /TR >
14: < /TABLE >

```

O *SSI* melhora a separação das camadas de negócio e apresentação, mas o *script* gerado por ele é responsável por toda a formatação dos resultados fazendo com que os *web designers* percam controle sobre a manipulação da aparência das páginas. O *ColdFusion* oferece uma abordagem de *templates* mais sólida, mas é um software proprietário e de

custo elevado. O valor de US\$ 8499,00 pela licença da versão Enterprise 2016 pode ser uma barreira para grande parte dos consumidores (ADOBE, 2016). O *Velocity* possui uma forma de programação que mistura comandos e funções do estilo *UNIX/Linux* com *XML*. O problema dessa mistura é que, além de gerar inconformidades com o padrão *XML*, também força os *web designers* a terem que conhecer não apenas *HTML* e *CSS*, mas também as funções do *Velocity*, algo que pode atrapalhar a divisão de trabalhos em equipes de Desenvolvimento de Softwares que possuam uma área de Desenvolvedores *Front-End*, que só cuidam da camada de apresentação (*HTML* e *CSS*), com a equipe de *Back-End*, que ficaria encarregada pela camada de negócios (SHKLAR; ROSEN, 2004).

Algoritmo 3 Exemplo de um *Hello World* em *Velocity*

```

1: < html >
2:   < body >
3:     #set($name = "World")
4:     Hello $name!
5:   < /body >
6: < /html >

```

Abordagens Híbridas combinam elementos de *script* com *templates*. Elas fornecem um poder de programação maior do que os *templates* “puros”, pois permitem a adição de blocos embutidos contendo *scripts*. Isso parece oferecer o benefício de uma estrutura orientada por página combinada com poder de programação adicional. Exemplos dessa abordagem incluem *PHP* e *Java Server Pages (JSP)*. No entanto, essas abordagens também possuem problemas na manutenibilidade das páginas, já que misturam as camadas de negócio com a camada de apresentação (SHKLAR; ROSEN, 2004).

O Algoritmo 4 apresenta a geração de uma página *web* dinâmica em *PHP*, que irá cumprimentar o usuário utilizando o parâmetro “name” recebido junto à requisição. É possível perceber que o código PHP é escrito dentro do código *HTML*.

Algoritmo 4 Exemplo de uma página em PHP

```

1: < html >
2:   < body >
3:     Hello <?php $_GET['name'];? >!
4:   < /body >
5: < /html >

```

2.2 Web frameworks

Um *framework* ou arcabouço, no contexto do desenvolvimento de *software*, é um conjunto de código pré-escrito ou bibliotecas que fornecem uma arquitetura e funcionalidades comuns a toda uma classe de aplicações. Tipicamente, um *framework* fornece uma estrutura pré-definida para a aplicação e pode ser visto como uma base sobre a qual uma aplicação pode ser construída. Um dos objetivos do uso de *web frameworks* é o reuso de uma arquitetura pré-definida, componentes e rotinas que são comumente utilizadas no desenvolvimento de uma aplicação *web* como, por exemplo, acesso a banco de dados e gerenciamento de sessões. O reuso de tal estrutura permite aumentar a produtividade no desenvolvimento das aplicações e fornecer maior confiabilidade às aplicações, uma vez que tornam possível a reutilização, abstraem detalhes de baixo nível e baseiam-se em estruturas amplamente utilizadas e testadas (SCHWARZ, 2013).

A maior parte dos *web frameworks* disponíveis hoje em dia são gratuitos (sob algum tipo de licenciamento *Apache*, *LGPL* e outros) e *open source*, o que significa que qualquer pessoa pode ler seu código e contribuir de alguma maneira, seja propondo melhorias, fazendo revisões/inspeções de código, ou identificando falhas de segurança (BJÖREMO; TRNINIĆ, 2010). Os *web frameworks* não são apenas utilizados para desenvolver aplicações *web* que dependem de interação do usuário com o navegador, mas também de *web services*, que permitem integrar aplicações e serviços. Os *web frameworks* geralmente são usados para o desenvolvimento de duas partes principais de uma aplicação *web* e são, portanto, classificados de acordo com seu propósito. Os *server-based frameworks* se concentram na programação do lado do servidor (ou *back-end*) das aplicações. Alguns exemplos desse tipo de *framework* são: *Struts* (BROWN; DAVIS; STANLICK, 2008), *Spring MVC* (DECK, 2008), e *CakePHP* (WATTS; GONZÁLEZ, 2014). Já os *client-based frameworks* tem o foco na programação da camada de apresentação e de interação com o usuário (*front-end*). Alguns exemplos desse tipo de *framework* são: *Bootstrap* (SPURLOCK, 2013), *Zurb Foundation* (HOREK, 2014) e *Materialize* (PRABHU; SHENOY, 2016).

A seguir, serão apresentados os detalhes de alguns *web frameworks*. O primeiro a ser apresentado é o *Java EE*, que embora seja tratado como uma especificação e não como um *web framework*, entendemos que devido às suas características e recursos fornecidos,

seria interessante apresentá-lo nesta seção. Além disso, ele foi a principal inspiração para a criação do *web framework* proposto neste trabalho.

Em seguida, são apresentados os *web frameworks* na linguagem C++, cuja existência se justifica em razão do objetivo de apoiar o desenvolvimento *web* usando a mesma linguagem.

2.2.1 *Java Enterprise Edition*

O *Java EE (Java Enterprise Edition)* é um conjunto de especificações para o desenvolvimento de, entre outras coisas, aplicações *web* e *web services* utilizando a linguagem *Java* (GONCALVES, 2013). No que se refere ao desenvolvimento de aplicações *web*, algumas especificações se destacam: *Servlet*, *JSP*, *JSTL* e *JSF*. Os *Servlets* atuam como minisservidores que podem ser instalados dentro do servidor principal, de modo a estender suas funcionalidades (HUNTER; CRAWFORD, 2001). A *JavaServer Pages (JSP)* funciona de maneira similar ao PHP, permitindo que o Java seja mesclado dentro de páginas *HTML* para a geração de páginas *web* dinâmicas. Internamente, o seu conteúdo será traduzido para um *Java Servlet*, sem que exista a necessidade de interferência do usuário (BASHAM; SIERRA; BATES, 2008). Devido às dificuldades geradas na manutenibilidade das aplicações feitas utilizando a *JSP*, a *Sun Microsystems* decidiu criar a *JavaServer Pages Standard Tag Library (JSTL)*, que também permite a geração de páginas *web* dinâmicas, mas substitui o código *Java* por tags *HTML*, de modo a facilitar a manutenibilidade das aplicações (SPIELMAN, 2003).

A *JavaServer Faces (JSF)* utiliza um modelo de interface de usuário baseada em componentes. Para isso, ela utiliza arquivos *XML* chamados *Facelets views*. As requisições são processadas pelo *FacesServlet*, que carrega a *Facelet view* adequada, constrói uma árvore de componentes, processa os eventos e devolve uma resposta ao cliente, normalmente uma página *HTML*. O estado dos componentes da interface do usuário e outros objetos de interesse são salvos ao final de cada requisição. Os objetos e estados podem ser salvos tanto no cliente, quanto no servidor (GEARY; HORSTMANN, 2010).

Em *Java EE*, diversas bibliotecas podem ser usadas para implementar diferentes funcionalidades e utilizar diferentes recursos em uma aplicação *web*. Exemplos de recursos e funcionalidades são a manipulação de código escrito em *JSON*, injeção de dependências,

separação de código de negócio e apresentação (*JSTL*) e acesso a banco de dados. Em geral, cada biblioteca tem diversas dependências e o usuário precisa escrever diversos arquivos de configurações para que a aplicação funcione corretamente. As aplicações escritas em *Java EE* devem rodar em servidores de aplicações como o *Apache Tomcat*, *Glassfish*, *IBM WebSphere*, entre outros. Os usuários devem configurar os apontamentos das *URL's* dos *servlets*, das bibliotecas e *frameworks* utilizados não só na aplicação, mas também no servidor utilizado.

O Algoritmo 5 apresenta uma pequena aplicação *web* feita com *Java EE*, utilizando *Servlets*, que irá cumprimentar o usuário. É possível notar que o *HelloServlet* foi mapeado para “/hello”, redefiniu o método *doGet* de *HttpServlet* e ao receber uma requisição, irá devolver uma página em *HTML* para o cliente.

Algoritmo 5 Exemplo de uma aplicação *web* em *Java EE*

```

1: package test;
2: import javax.servlet.http.HttpServlet;
3: import javax.servlet.http.HttpServletRequest;
4: import javax.servlet.http.HttpServletResponse;
5: import javax.servlet.annotation.WebServlet;
6: @WebServlet("/hello")
7: public class HelloServlet extends HttpServlet{
8:     public void doGet (HttpServletRequest req, HttpServletResponse res)
9:         throws ServletException, IOException{
10:         res.getWriter().write("< html >< body > Hell world! < /body >< /html >"
11:         );
12:     }

```

Apesar das facilidades oferecidas por ferramentas como *Maven*, *Ant* e *Gradle* para gerenciar projetos complexos como os *Java EE*, desenvolver, configurar e manter todas as bibliotecas e *frameworks* atualizados, e interoperando em uma aplicação *Java EE*, é um desafio para uma grande parte dos programadores, mesmo os mais experientes (SINHA; KARIM; GUPTA, 2015; SWAIN et al., 2016).

2.2.2 Wt: Web Tool Kit

O *Wt* permite que os programadores criem aplicações para a *web* utilizando a linguagem C++. Ele transforma os objetos de C++ em *HTML* para criar uma página *web*, no lado servidor, e a envia para o navegador (DUMON; DEFORCHE, 2008). Entretanto,

o *Wt* induz o programador a criar quase tudo em C++, até mesmo botões em *HTML* são criados em forma de objetos C++. O resultado direto disso é que, em muitos casos, até mesmo para se corrigir um pequeno texto no *HTML* será necessário alterar código em C++ e recompilá-lo. O Algoritmo 6 mostra a inicialização do servidor *web*.

Algoritmo 6 Iniciando o servidor *web* *Wt*

```

1: #include "helloapplication.h"
2: Wt :: WApplication * createApplication(const Wt :: WEnvironment& env)
3: {
4:     return new HelloApplication(env);
5: }
6: int main(int argc, char **argv)
7: {
8:     return Wt :: WRun(argc, argv, &createApplication);
9: }
```

O Algoritmo 7 mostra a criação de uma *WApplication*. É possível observar que o código *HTML* é todo gerado em C++.

Algoritmo 7 Criando uma *WApplication* uma aplicação em *Wt: Web Tool Kit*

```

1: class HelloApplication : public Wt :: WApplication
2: {
3:     Wt :: WLineEdit * nameEdit;
4:     Wt :: WText * greeting;
5: public :
6:     void greet()
7:     {
8:         greeting_ > setText("Hello," + nameEdit_ > text());
9:     }
10:     HelloApplication(const Wt :: WEnvironment& env) : Wt :: WApplication(env)
11:     {
12:         setTitle("Helloworld");
13:         root() -> addWidget(new Wt :: WText("Digite seu nome :"));
14:         nameEdit = new Wt :: WLineEdit(root());
15:         Wt :: WPushButton * button = new Wt :: WPushButton("Enviar.", root());
16:         root() -> addWidget(new Wt :: WBreak());
17:         greeting_ = new Wt :: WText(root());
18:         button -> clicked().connect(this, &HelloApplication :: greet);
19:     }
20: };
```

2.2.3 *QtWebApp*

O *QtWebApp* permite que os programadores criem aplicações *web* em C++ e como o nome sugere, ele utiliza o *Qt Framework* em seu desenvolvimento. Atualmente, esse *web framework* disponibiliza um servidor de aplicações, *Controllers (Servlets)* para lerem e responderem requisições *HTTP/S*, e utiliza uma ferramenta chamada *Templates* que consiste em um conjunto limitado de “tags” para separar código C++ de código *HTML* e gerar páginas *web* dinâmicas (FRINGS, 2010). No entanto, essas “tags” tiram a estética do *HTML*, além de ser necessário pré-processá-las dentro do código C++. O Algoritmo 8 apresenta uma pequena aplicação *web* feita utilizando o *QtWebApp*, que irá construir uma página *web* dinâmica contendo todos os parâmetros *HTTP* enviados junto à requisição e, posteriormente, enviá-la para quem a requisitou. É possível notar que o código da página dinâmica precisa ser pré-processado dentro da aplicação C++.

Algoritmo 8 Exemplo de uma aplicação em *QtWebApp*

```
void TemplateController :: service(HttpRequest& request, HttpResponse& response)
{
    response.setHeader("Content - Type", "text/html; charset = ISO - 8859 - 1");
    Template t = templateCache -> getTemplate("demo", request.getHeader("Accept -
Language"));
    t.enableWarnings();
    t.setVariable("path", request.getPath());
    QMap < QByteArray, QByteArray > headers = request.getHeaderMap();
    QMapIterator < QByteArray, QByteArray > iterator(headers);
    t.loop("header", headers.size());
    int i = 0;
    //Processando as "tags" do HTML manualmente.
    while(iterator.hasNext())
    {
        iterator.next();
        t.setVariable(QString("header%1.name").arg(i), QString(iterator.key()));
        t.setVariable(QString("header%1.value").arg(i), QString(iterator.value()));
        ++ i
    }
    response.write(t.toLatin1(), true);
}
```

O Algoritmo 9 mostra o uso das “tags” do *QtWebApp* dentro de uma página *HTML*.

Algoritmo 9 Exemplo de uma página em *QtWebApp*

```

1: < html >
2:   < body >
3:     Hello,< br > you requested the path : {path} < p >
4:     And your web browser provided the following headers :< p >
5:     {loop header}
6:     < b > {header.name} :< /b > {header.value} < br >
7:     {end header}
8:   < /body >
9: < /html >

```

2.2.4 POCO C++ Libraries

A *POCO C++ Libraries* fornecem bibliotecas para acesso a banco de dados, manipulação de *JSON* e *XML*, compressão de arquivos e *FileSystem*. Um dos recursos mais notáveis da *POCO C++ Libraries* é uma ferramenta chamada *CSP*, que é a versão *JSP* para C++ (OBILTSCHNIG, 2005). Porém, assim como a *JSP*, a *CSP* fornece vários recursos para que o programador misture a camada de negócios (C++) com a camada de apresentação (*HTML*), o que pode gerar problemas na manutenibilidade das páginas e impedir a entrada de *web designers* (OBILTSCHNIG, 2005).

O Algoritmo 10 apresenta uma pequena aplicação *web* feita com *POCO C++ Libraries*. É possível notar que o código C++ está presente dentro do *HTML*.

Algoritmo 10 Exemplo de uma página *web* utilizando *POCO C++ Libraries*

```

1: < %!
2:   #include "iostream"
3: % >
4: < %
5:   std :: string hello("Hello World");
6: % >
7: < html >
8:   < head >
9:     < title > Hello World < /title >
10:   < /head >
11:   < body >
12:     < p >< % = hello% >< /p >
13:   < /body >
14: < /html >

```

2.2.5 Crow

O *Crow* é baseado no *microframework Flask*, um *microframework* feito em *Python* e que fornece recursos básicos para o desenvolvimento de pequenas aplicações *web*. Por ser um *microframework*, o *Crow* oferece recursos bastante limitados. Além disso, o *Crow* exige a instalação da *Boost*, uma biblioteca de C++ para suprir alguns recursos inexistentes na linguagem C++. Entretanto, a *Boost* não é totalmente multiplataforma, como o *Qt Framework*, mas fornece suporte ao *Windows*, *Linux* e *Mac*.

O Algoritmo 11 apresenta uma pequena aplicação *web* feita com *Crow*.

Algoritmo 11 Exemplo de uma aplicação *web* em *Crow*

```

1: #include "crow.h"
2: int main()
3: {
4:     crow :: SimpleApp app;
5:     CROW_ROUTE(app, "/")([]){return "Hello world";};
6:     app.port(18080).multithreaded().run();
7: }
```

2.3 Análise de desempenho

Existe uma série de trabalhos encontrados na literatura que avaliam e comparam o desempenho de diferentes conjuntos de linguagens de programação, *frameworks* e funcionalidades.

Scott comparou o desempenho do *PHP* e *JSP* nos servidores *web* mais populares, *Apache* e *Lighttpd*, e concluiu que A *JSP* tem um desempenho melhor do que *PHP* (TRENT et al., 2002). Ramana analisou as diferenças de desempenho entre *PHP* e linguagens compiladas como a linguagem C, apontando o baixo desempenho do *PHP* (RAMANA; PRABHAKAR, 2005). Hundt implementou alguns algoritmos nas linguagens C++, Java, Scalla e GO, e chegou à conclusão de que o C++ possui o melhor desempenho (HUNDT, 2011).

Björemo utilizou seis critérios (documentação e aprendizagem, convenção sobre configuração, ambiente de desenvolvimento integrado, internacionalização, entrada de dados do usuário para validação e teste) para avaliar os *frameworks* *CakePHP*, *Grails*, *Ruby on Rails*, *Listras*, *Spring Roo* e *Wicket*, e as conclusões desta avaliação, nesses

aspectos, foram de que não há nenhum *framework* superior e que não se deve aprender uma nova linguagem de programação apenas para usar um *web framework* recomendado (BJÖREMO; TRNINIĆ, 2010), pois todos são muito parecidos.

Prechelt, comparou o tempo de implementação para o desenvolvimento em linguagens de *script* (*Perl*, *Python*, *Rexx* e *Tcl*) com o tempo para a programação da mesma funcionalidade em *C*, *C++* e *Java*. Ele concluiu que o tempo de desenvolvimento das aplicações em linguagens de *script* é significativamente menor, cerca de metade do tempo. (PRECHELT, 2003).

Fourment e Gillings compararam o uso de memória e a velocidade de execução para três métodos padrões de bioinformática, implementados em seis linguagens de programação diferentes (*C*, *C++*, *C#*, *Java*, *Perl* e *Python*), e chegaram à conclusão de que as implementações feitas em *C* e *C++* são as mais rápidas e consomem uma menor quantidade de memória (FOURMENT; GILLINGS, 2008). Prechelt criou 80 implementações com o mesmo conjunto de requisitos nas linguagens *C*, *C++*, *Java*, *Perl*, *Python*, *Rexx*, e *Tcl* para compará-las. Os resultados indicaram que as linguagens de *script* *Perl*, *Python*, *Rexx* e *Tcl* são mais produtivas do que linguagens convencionais. Entretanto, em termos de tempo de execução e consumo de memória *C*, *C++* e *Java* se saíram melhor (PRECHELT, 2000).

Neves, Nuno e Durães compararam as linguagens *Java* e *PHP*, e seus resultados sugeriram que o *Java* é mais escalável e seguro que o *PHP* (NEVES; PAIVA; DURÃES, 2013).

Como pode ser visto, existe uma série de estudos na área voltados a verificar o desempenho de linguagens e *web frameworks*. De um modo geral, as aplicações feitas em linguagens compiladas, como *C* e *C++* possuem um desempenho melhor e um menor consumo de recursos computacionais para se realizar a mesma tarefa. Entretanto, a maior parte desses estudos são relativamente antigos, não envolvem o mesmo conjunto de ferramentas que serão utilizadas nesse trabalho e nem as mesmas versões.

2.4 Considerações finais

Existem diversos *web frameworks* disponíveis para o desenvolvimento de aplicações e serviços *web*. Os *web frameworks* mais robustos, e que oferecem apoio a funcionalidades e recursos complexos, são, em geral, difíceis de usar e configurar até mesmo para desenvolvedores experientes (SINHA; KARIM; GUPTA, 2015; SWAIN et al., 2016). Alternativas mais simples que, foram verificadas neste estudo, geralmente falham em manter a camada de negócio e a camada de apresentação separadas, além de requerer que o desenvolvedor desenvolva a maior parte das conexões entre componentes e uso de outras bibliotecas e recursos. Além disso, o desempenho das aplicações pode ser prejudicado em razão da necessidade de interação entre vários componentes, alto consumo de recursos computacionais e a predominância de linguagens interpretadas, pois são mais lentas do que linguagens compiladas (LUBIN; DUNNING, 2015).

Entre as opções pesquisadas, um mecanismo eficiente para se manter a camada de negócios separada da camada de apresentação e ao mesmo tempo manter um bom desempenho, sem *overhead* na criação de processos, é por meio do uso dos *Servlets/Controllers* em conjunto com classes auxiliares e *JSTL* do *Java*. Em C++, a *POCO C++ Libraries* e o *QtWebApp* carecem de um mecanismo como a *JSTL* para separar a camada de apresentação da camada de negócios. No entanto, para se criar algo como a *JSTL* é necessário utilizar metaprogramação, como *reflection*, mas a linguagem C++ não oferece essa técnica nativamente em seu padrão oficial *ISO* (CHOCHLIK, 2012). Isso pode gerar dificuldades para implementar algo como a *JSTL*, mas é possível suprir essa ausência utilizando o *MetaObject Compiler (moc)* do *Qt* (EZUST; EZUST, 2006), por exemplo.

3 CWF - C++ Web Framework

Neste capítulo, é apresentado o *CWF (C++ Web Framework)*, um *web framework* proposto neste projeto de pesquisa como uma alternativa às opções disponíveis no mercado e que pretende amenizar os problemas relatados na literatura a respeito do desenvolvimento *web*. Em primeiro lugar, é apresentada uma lista de requisitos mínimos esperados de uma aplicação *web* para os quais os *web frameworks* devem dar apoio. Em seguida, são apresentadas as decisões de projeto tomadas na implementação do *framework* proposto de tal forma que os problemas relatados na literatura sejam mitigados proposto de tal forma que os problemas relatados na literatura sejam mitigados e os requisitos sejam, ao mesmo tempo, atendidos. Por fim, o *CWF* é apresentado em detalhes.

3.1 Requisitos mínimos

Para se criar uma aplicação web é necessário um servidor *TCP/IP* que receba requisições no protocolo *HTTP (Hiper Text transfer Protocol)*, interprete essa requisição e devolva uma resposta ao cliente utilizando o mesmo protocolo. Entretanto, é bastante comum que os *web frameworks* também forneçam serviços adicionais, como:

1. Gerenciamento de sessões (para gerenciamento de *logins*, por exemplo);
2. Filtros ou interceptadores (ideal para ser colocado em condições de pré-validação comuns a todos os *Controllers*);
3. *HTTPS* (para conexões seguras utilizando criptografia);
4. Acesso a bancos de dados;
5. *ORM* (para mapeamento de classes para tabelas no banco de dados);
6. Manipulação de arquivos *XML* e *JSON* (para a utilização em requisições de *web services*, por exemplo);
7. Criação de *web services*;
8. Forcem o desenvolvedor a seguir alguma arquitetura, como *MVC*, por exemplo.

3.2 Decisões de projeto

As decisões de projeto tomadas para a implementação de um *web framework* podem exercer grande influência na aplicação *web*. Portanto, baseado nos estudos realizados a respeito dos problemas que normalmente são apresentados nos *web frameworks* e na lista de requisitos mínimos apresentados anteriormente, algumas decisões foram tomadas para minimizar os problemas recorrentes e, ao mesmo tempo, fornecer flexibilidade e bom desempenho às aplicações desenvolvidas com o *CWF*. De forma resumida, decidiu-se desenvolver um *web framework* que forneça apoio à criação de aplicações escritas na linguagem de programação C++, em conjunto com as bibliotecas fornecidas pelo *Qt Framework*. Além disso, decidiu-se que cada aplicação *web* deve funcionar como um servidor isoladamente. Nas próximas subseções, cada uma dessas decisões é descrita em detalhes.

3.2.1 Linguagem de programação

Optou-se por desenvolver um *web framework* que apoia a implementação de aplicações *web* na linguagem de programação C++ por diversos motivos:

1. **Tempo de resposta e consumo de recursos computacionais (memória *RAM* e Processador):** uma ampla pesquisa bibliográfica revelou que a linguagem C++ apresenta, de forma consistente, um bom desempenho quanto ao tempo de resposta e consumo de recursos computacionais em diversos testes realizados. Ramana analisou as diferenças de desempenho entre PHP e linguagens compiladas como a linguagem C, apontando o baixo desempenho do PHP (RAMANA; PRABHAKAR, 2005). Hundt implementou alguns algoritmos nas linguagens C++, Java, Scala e GO, e chegou a conclusão de que o C++ possui o melhor desempenho (HUNDT, 2011). O site *The Computer Language Benchmarks Game* realiza constantemente testes de desempenho com 27 diferentes linguagens de programação, os testes normalmente são reexecutados quando é liberada uma nova versão de linguagem de programação, máquina virtual ou compilador, e, nesses testes, as linguagens C e C++ quase sempre vencem as demais linguagens nos quesitos de desempenho (GAME, 2018).
2. **Público alvo:** um estudo da *JetBrains*, publicado em 2015, a respeito das linguagens de programação mais utilizadas, e o número de programadores estimado de cada

uma, mostrou que existem, aproximadamente, 4.4 milhões de desenvolvedores C++ espalhados ao redor do mundo e aproximadamente 1.9 milhão de desenvolvedores C (JETBRAINS, 2015). Além disso, as linguagens C e C++ estão no top 4 do TIOBE *Index*, de Agosto de 2019, que mede as linguagens de programação mais utilizadas no mundo (TIOBE, 2019).

3. **Atualização de versões e retrocompatibilidade com versões anteriores:** a atualização de versões é um fator importante para a escolha, seja de uma linguagem de programação, biblioteca, *framework*, *software* e etc. As atualizações são importantes para manter as linguagens com recursos suficientes para suprir as demandas das novas aplicações, mas, muitas vezes, existem atualizações que causam incompatibilidade com versões anteriores. Ao analisar a linguagem C++, é possível constatar que ela passou a ser atualizada a cada três anos, desde 2011, e que os novos padrões lançados não geram incompatibilidade com as versões anteriores. Até o momento, já foram lançadas as seguintes atualizações para a linguagem: 98, 03, 11, 14 e 17. Atualmente, o comitê *ISO* está finalizando o padrão C++ 20 e já possui planos para o C++ 23 e C++ 26.

Existem outros fatores associados à escolha da linguagem C++. Por exemplo, em seu padrão 11, foram inseridos os ponteiros inteligentes, que resolvem definitivamente os problemas relacionados a vazamentos de memória, que antes eram muito comuns no C++.

3.2.2 Framework de apoio

Diversos recursos utilizados por aplicações tradicionais (*stand alone* ou *desktop*) não são necessariamente fornecidos de forma nativa pelas linguagens de programação. Em geral, um *framework* de apoio ou um conjunto de bibliotecas é utilizado para que o processo de desenvolvimento seja facilitado. Até a sua versão mais recente, a linguagem C ou C++ não oferece nativamente alguns recursos como reflexão (*reflection*), acesso a banco de dados, manipulação de arquivos *JSON* e *XML*, *sockets*, dentre outros. Portanto, uma decisão importante na implementação de um *web framework* também é a escolha do *framework* subjacente ou as bibliotecas que serão utilizadas para apoiar a implementação das aplicações.

Foi feita uma pesquisa inicial para verificar qual seria a melhor biblioteca de propósito geral ou *framework* para auxiliar na criação do novo *web framework*. A *boost* e o *Qt Framework* são as opções mais conhecidas presentes atualmente no mercado (GOOGLE, 2016). Entretanto, a *boost* só apoia o desenvolvimento para *Windows*, *Linux* e *Mac*. Além disso, muitos dos recursos presentes na *boost*, que seriam atrativos, já se encontram dentro dos padrões oficiais da linguagem C++, como é o caso das *threads*, *chrono*, ponteiros inteligentes, dentre outros recursos. Outros dois fatores que nos desmotivaram de utilizar a *boost* foram: ausência de classes para se fazer acesso a bancos de dados e documentação escassa e limitada.

O *Qt*, um *framework* multiplataforma para desenvolvimento de aplicações em C++, foi escolhido para o desenvolvimento do *CWF* por diversos motivos:

1. **Áreas de aplicação:** o *Qt Framework* fornece todas as ferramentas necessárias para que se possa criar um *web framework*. Além disso, o *Qt Framework* é muito amplo, atendendo muitas áreas de aplicação que irão resolver o *dependency hell* para muitos usuários e minimizá-lo para outros, pois o *Qt Framework* fornece 14 módulos que são multiplataforma e mais de 40 *add-ons* que, em sua maioria, são multiplataforma. Embora o *Qt Framework* forneça ferramentas para vários tipos de aplicação, os seguintes recursos foram cruciais para sua adoção: *sockets*, classes para manipulação de *XML* e *JSON*, acesso a banco de dados e *reflection*. Desta forma, se o desenvolvedor assim desejar, nenhuma biblioteca externa deverá ser usada no desenvolvimento de sua aplicação. Entretanto, se houver interesse, o desenvolvedor pode utilizar bibliotecas externas para a implementação dos recursos desejados.
2. **Atualização de versões:** embora não haja uma data específica definida para o lançamento de cada versão do *Qt Framework* no site da *Qt Company*, empresa responsável pelo *Qt Framework*, ao analisar o repositório com todas as versões do *Qt Framework* (da versão 1 à versão 5.13), nota-se que o *Qt Framework* tipicamente lança entre 1 a 3 novas versões todo ano, e normalmente as versões não causam incompatibilidades entre si, garantindo que o código anterior continuará funcionando na próxima versão.
3. **Plataformas apoiadas:** o *Qt Framework* é multiplataforma e permite que o mesmo código criado para um sistema operacional seja exatamente o mesmo para outro sistema operacional, bastando recompilar o código. Os sistemas operacionais para

desktop suportados pelo *Qt Framework* são: *Windows*, *Linux* e *Mac*. Para plataformas *mobile*: *Android*, *iOS*, *tvOS*, *watchOS* e *WinRT*. Além disso, o *Qt Framework* também funciona para os seguintes sistemas embarcados: *Embedded Linux*, *INTEGRITY*, *QNX* e *VxWorks*.

4. **Aplicações híbridas:** o *Qt* permite a criação de aplicativos híbridos, que são tanto *desktop* como aplicações *web*, como é o caso dos *progressive webapps*. Um desenvolvedor pode criar um banco de dados na nuvem e um *web service* para receber informações de telefones de contatos, por exemplo. Em seguida, o desenvolvedor pode criar uma aplicação *mobile* em *Qt*, utilizando *QML* ou *QtWidgets* e um banco de dados *SQLite*. Quando o usuário estiver em uma área sem conexão com a internet e precisar salvar um contato, esse contato é salvo no banco de dados *SQLite*, um banco de dados *SQL* local. Quando o usuário obtiver uma conexão com a internet, esses dados podem ser sincronizados com o banco de dados da nuvem por meio do *web service* feito em *CWF*.
5. **Presença no mercado:** atualmente, segundo dados da *Qt Company*, o *Qt Framework* é utilizado por mais de 1 milhão de usuários ao redor do mundo (COMPANY, 2019). Além disso, também segundo a *Qt Company*, o *Qt Framework* é utilizado por 8 das 10 maiores empresas do mundo listadas no *Fortune 500* (COMPANY, 2019). O *Qt Framework* também possui certificação *IEC 62304* e *ISO 14971*, que certifica que o *Qt Framework* é seguro para a criação de aplicações voltadas para a medicina e *RTOS* (COMPANY, 2019).
6. **Licenciamento:** o *Qt Framework* atualmente é disponibilizado em duas diferentes versões: a comercial e a *open source*, disponibilizada sob licenciamento *(L)GPL v3*. A versão comercial fornece mais recursos e ferramentas do que a versão *open source*, além de fornecer suporte técnico, caso o usuário necessite de algum auxílio. Entretanto, a versão *open source* possui todos os recursos necessários para se criar um *web framework*.
7. **Documentação:** o *Qt Framework* possui uma documentação extensa e detalhada, o que facilita sua utilização tanto na criação do *CWF* quanto na criação das aplicações baseadas no *CWF*.

Apesar do *CWF* ter sido construído usando o *Qt*, os usuários também podem utilizar a *boost* e/ou qualquer outra biblioteca no desenvolvimento das suas aplicações *web*,

podendo fazer uma mesclagem entre: *Qt Framework*, *boost*, bibliotecas próprias e outras bibliotecas. Entretanto, a ideia do *CWF* é fazer com que o usuário elimine ou pelo menos minimize o número de dependências necessárias para criar suas aplicações.

3.2.3 Arquivos de configuração

Um dos fatores que dificultam, e muito, o desenvolvimento de aplicações *web* são os arquivos de configuração (SINHA; KARIM; GUPTA, 2015; SALAS-ZARATE et al., 2015; VUORIMAA et al., 2016). Para tentar reduzir essa dificuldade, optou-se por criar um único arquivo de configuração, no formato “.ini”, de modo que todas as suas 19 opções de configuração já estão expressas diretamente no arquivo em um formato chave/valor. Desse modo, os desenvolvedores não terão que pesquisar em documentações quais são as opções disponíveis para as configurações e o formato chave/valor torna a sua alteração bastante simplificada.

3.2.4 Separação entre código de apresentação e negócios

Um dos grandes problemas encontrados no desenvolvimento de aplicações *web* é a constante mistura da camada de apresentação (código *HTML* e *CSS*) com a camada de negócios (código de alguma linguagem de programação). Essa mistura gera problemas de manutenibilidade na aplicação *web* e também atrapalha a vida dos *web designers*, que normalmente só possuem conhecimento sobre a camada de apresentação e não sobre a camada de negócios (SHKLAR; ROSEN, 2004). Ao analisarmos as diferentes formas de desenvolvimento de aplicações *web* no capítulo 2, chegamos à conclusão de que a abordagem da *JSTL* (*JavaServer Pages Standard Tags Library*) seria a melhor opção, pois ela não permite código de linguagem de programação dentro da camada de apresentação e, para fazer a exibição de conteúdos, ela cria uma série de novas tags. Deste modo, não há mistura entre a camada de negócios com a camada de apresentação, e o *HTML* não perde sua formatação. Com base na *JSTL*, criamos a *CSTL* (*C++ Server Pages Standard Tags Library*), que será explicada mais adiante.

3.2.5 Contêiner de aplicação

Para se construir uma aplicação *web*, é necessário ter um servidor *web* rodando em alguma máquina para responder às requisições. Para fazer o *CWF* rodar dentro de algum servidor *web* já conhecido, seria necessário fazer com que ele fosse implementado sob algum protocolo, como *CGI* ou *FastCGI*, por exemplo, o que poderia prejudicar seu desempenho. Além disso, o usuário teria que fazer o *download*, instalar e configurar algum servidor com suporte a um desses protocolos para fazer o *CWF* rodar dentro de um desses servidores, algo que não é uma tarefa simples, principalmente para iniciantes. Outro fator é que a ideia do *CWF* é ser multiplataforma, de modo que o usuário possa transformar até mesmo o seu *smartphone* com *Android* ou *iOS* em um servidor *web* e a maior parte dos servidores *web* rodam apenas em *Windows*, *Linux* e *Mac*. Os fatores de ter que adotar um protocolo que poderia prejudicar o desempenho do *CWF*, a dificuldade extra que seria acrescentada para os usuários para baixar, instalar e configurar o servidor *web*, e o fato de que a maior parte dos servidores não são totalmente multiplataforma, motivou a criação de um servidor próprio para o *CWF*.

3.2.6 Inspiração no Java Servlets

Como já mencionado no Capítulo 2, as abordagens de desenvolvimento baseadas em protocolos como *CGI*, por exemplo, causam uma grande perda de desempenho e um alto consumo de recursos computacionais (memória *RAM* e Processamento), pois, a cada nova requisição ao servidor, ele configura uma série de parâmetros em várias variáveis de ambiente e, em seguida, sobe uma aplicação inteira para responder à requisição e o *output* da aplicação é direcionada para o servidor, induzindo o usuário a misturar as camadas de negócio e apresentação. O *FastCGI* melhora um pouco essa abordagem mantendo um certo número de aplicações sendo executadas continuamente. Os *Java Servlets* adotam uma postura diferente dos protocolos *CGI* e *FastCGI*, pois, ao invés de configurar uma série de variáveis de ambiente, e, em seguida, executar uma aplicação inteira, eles simplesmente disparam um novo *thread*(tarefa) para atender à requisição, reduzindo o consumo de memória, processamento e aumentando a velocidade de resposta

às requisições. Os *Controllers* do *CWF* foram inspirados nos *Java Servlets* para trazer esses mesmos benefícios ao novo *web framework*.

3.3 Descrição do C++ Web Framework (CWF)

O *C++ Web Framework (CWF)* é um *web framework MVC* criado para auxiliar o desenvolvimento de aplicações *web* usando a linguagem C++ em conjunto com o *Qt Framework*, sendo inspirado no *Java Servlets* para evitar o alto consumo de memória *RAM* e processamento de abordagens como *CGI* e *FastCGI*, e também na *JSTL*, de modo a oferecer uma melhor separação entre a camada de negócios da camada de apresentação. O *CWF* pode ser especialmente atraente para os desenvolvedores C++ que desejam desenvolver aplicações *web*, mas não desejam aprender uma nova linguagem de programação ou lidar com *frameworks* complexos com diversos componentes e arquivos de configuração, e com extensa dependência de bibliotecas externas.

Uma das características mais interessantes do *CWF* é que ele combina o poder da linguagem C++ com a flexibilidade do *Qt Framework*. Os desenvolvedores podem utilizar recursos do *Qt Framework* para acessar bancos de dados, manipular imagens, manipular *JSON* e *XML*, criar testes de unidade e muito mais. Assim, as aplicações *web* desenvolvidas neste ambiente tendem a apresentar as seguintes características:

1. Alto desempenho, uma vez que não utilizam linguagens interpretadas;
2. Baixo consumo de recursos computacionais (memória e processamento);
3. Um mecanismo simples de configuração;
4. Páginas *web* dinâmicas criadas com a *CSTL (C++ Server Pages Standard Tag Library)*, que será descrita em detalhes mais à frente no texto;
5. Separação da camada de apresentação (*HTML*, por exemplo) da camada de negócios (código *back-end*);
6. Multiplataforma.

O *framework* proposto neste projeto de pesquisa está descrito em detalhes nas próximas sessões, em que sua arquitetura é apresentada juntamente com uma explicação sobre cada um de seus componentes. Além disso, são apresentados exemplos de templates e códigos utilizados para instanciar uma aplicação *web* simples utilizando a estrutura do *CWF*.

3.3.1 A arquitetura do *CWF*

A Figura 2 mostra uma visão geral da arquitetura do *CWF*. Uma aplicação web feita no *CWF* é construída na linguagem C++ e precisa utilizar o *Qt Framework* e suas bibliotecas para suprir algumas áreas que a linguagem C++, nativamente, não fornece suporte, como é o caso do acesso a banco de dados, manipulação de *JSON* e *XML*, rede, *Reflection*, dentre outros. Uma característica particular do *CWF* é que a aplicação tem seu próprio servidor *web* privado, o *CPPWebServer*, o que significa que o desenvolvedor não precisa implantar sua aplicação em um servidor web como o *Apache*, por exemplo. Isto é muito útil para manter a infra-estrutura simples e fácil de se configurar. Os *Controllers* atuam como minisservidores que podem ser instalados dentro do *CPPWebServer* para estender sua funcionalidade, respondendo às requisições e apresentando conteúdo dinâmico por meio de páginas *web* e *CSTL* (*C++ Server Pages Standard Tag Library*). Os *controllers* também podem usar classes regulares do C++. Uma visão geral da estrutura de classes do *framework* pode ser vista na Figura 3.

Figura 2 – Arquitetura do *CWF*

Fonte: Herik Lima de Castro

Figura 3 – Diagrama de Classes

Fonte: Herik Lima de Castro

Embora o CWF utilize as bibliotecas do Qt, o uso da IDE Qt Creator não é obrigatório, podendo ser utilizado em qualquer outra IDE de C++.

3.3.2 A estrutura de um projeto CWF

Para criar aplicações utilizando o *CWF*, os desenvolvedores devem ter o *Qt Framework* e as bibliotecas do *CWF* instaladas na máquina. Toda aplicação que utilize o *CWF*, obrigatoriamente, irá usar os módulos *core*, *network* e *xml* do *Qt Framework*, e, se fizer acesso a banco de dados, será necessário adicionar o módulo *sql* também. A Figura 4 mostra como fica a hierarquia de pastas de um projeto *CWF*: **HelloWorld** é a pasta principal do projeto; **HelloWorld.pro** é o arquivo de projeto do *Qt Framework*; **sources** mantém todos os arquivos de código fonte do projeto; **Other files** mantém arquivos auxiliares do projeto; **server** é uma pasta obrigatória para todo projeto *CWF*; **config** mantém o arquivo de configuração do projeto; **cppwebserverpages** mantém as páginas default da aplicação (`index.html`, `403 and 404`, por exemplo); **log** mantém informações sobre o log do servidor; e **ssl** mantém os arquivos de configuração para o protocolo *HTTPS*.

Figura 4 – A estrutura de pastas de um projeto *CWF*

Fonte: Herik Lima de Castro

O arquivo `CPPWeb.ini` é responsável por manter informações sobre a configuração do *web server*, que é único para cada aplicação. Um exemplo de um arquivo de configuração de um *web server* do *CWF* é apresentado no Algoritmo 12. Nesse arquivo, `host` e `port` configuram o *IP* e o número de porta que o servidor vai utilizar; `maxThread` configura a quantidade máxima de *threads* que o servidor vai criar; `cleanupInterval` configura o período para limpeza de cache; `timeOut` configura o tempo máximo de vida de *threads* que não estão sendo utilizados; `sessionExpirationTime` configura o tempo padrão para expiração de sessões; `maxUploadFile` configura o tamanho máximo permitido para as requisições; `maxLogFile` configura o tamanho máximo do arquivo de *log*; `sslKeyFile` aponta para o arquivo `.key` e `sslCertFile` aponta para o arquivo `.cert`, que estão relacionados com certificações digitais; `sslPassPhrase`, `sslKeyAlgorithm`, `sslEncodingFormat`, `sslKeyType`, `sslPeerVerifyMode` e `sslProtocol` estão relacionados com configurações das certificações digitais; `indexPage` configura a página principal do projeto a ser exibida; `accessCPPWebIni` informa se o arquivo `CPPWeb.ini` pode ser acessado remotamente; `accessServerPages` informa se a aplicação usará as páginas padrão do *CWF*. O `CPPWeb.ini` foi projetado para ser o mais simples possível, de modo a facilitar o desenvolvimento e manutenção das aplicações.

3.3.3 As classes `CppWebApplication` e `CppWebServer`

A classe `CppWebApplication` é a principal classe de uma aplicação web usando *CWF*. Ela encapsula o `CppWebServer` e outras classes importantes do *Qt Framework*, como

Algoritmo 12 Arquivo de configuração do *CWF* (CPPWeb.ini)

```

1: [config]
2: host = Any
3: port = 8080
4: maxThread = 200
5: cleanupInterval = 60000
6: timeOut = 60000
7: sessionExpirationTime = 6000
8: maxUploadFile = 20971520
9: maxLogFile = 20000000
10: ; sslKeyFile = /config/ssl/key.pem
11: ; sslCertFile = /config/ssl/cert.pem
12: ; sslPassPhrase = cppwebframework
13: ; sslKeyAlgorithm = rsa
14: ; sslEncodingFormat = pem
15: ; sslKeyType = privatekey
16: ; sslPeerVerifyMode = verifynone
17: ; sslProtocol = tlsv1ssl3
18: indexPage = /config/cppwebserverpages/index.view
19: accessCPPWebIni = false
20: accessServerPages = true

```

a `QCoreApplication` que é responsável por fornecer um laço de eventos para aplicações *Qt* que não usam interface gráfica. Esta classe é a responsável pela inicialização da aplicação. O Algoritmo 13 mostra uma *view* que será posteriormente chamada pelo *Controller* “HelloWorldController”.

Algoritmo 13 *View helloworld*

```

1: < html >
2:   < body >
3:     Hello World!
4:   < /body >
5: < /html >

```

Um exemplo de como instanciar uma `CppWebApplication` é apresentado no Algoritmo 14. A `CppWebApplication` deve receber os argumentos `argc` e `argv` da função `main` em seu construtor, e também o caminho para a pasta `server`. Se a pasta `server` não for encontrada no caminho especificado no construtor, ele irá procurar por ela dentro da pasta onde a aplicação executável se encontra. Observe também que o método `start` é chamado para iniciar o servidor. Para verificar se o servidor foi iniciado de maneira correta, o desenvolvedor pode consultar o arquivo de `log` ou acessar o endereço especificado no arquivo de configuração (`host:port`). O Algoritmo 14 também apresenta a classe “Hel-

loWorldController” que herda de Controller e reescreve seu método `doGet`, de modo que as requisições HTTP enviadas via método GET, para esse Controller, irão receber como resposta a view “helloworld”. Repare que a classe “HelloWorldController” é mapeada para o servidor com a URL “/hello”.

Algoritmo 14 Criando uma aplicação *web* em *CWF*

```

1: #include < cwf/cppwebapplication.h >
2: class HelloWorldController : public CWF :: Controller
3: {
4: public :
5:     void doGet(CWF :: Request &request, CWF :: Response &response)
6:     const override
7:     {
8:         request.getRequestDispatcher("/pages/helloworld.view").
9:             forward(request, response);
10:    }
11: };
12: int main(int argc, char * argv[])
13: {
14:     CWF :: CppWebApplication server(argc, argv, "/server");
15:     server.addController < HelloWorldController > ("/hello");
16:     return server.start();
17: }
```

A Figura 5 mostra o resultado de uma requisição a “http://localhost:8080/hello”.

Figura 5 – Resultado da requisição

Fonte: Herik Lima de Castro

Cada aplicação possui seu próprio *web server* e é possível colocar várias instâncias em execução dentro de uma mesma máquina mudando apenas o número da porta. Pelo fato dos arquivos-fonte do *CWF* serem compilados dentro de uma biblioteca dinâmica, todas as instâncias de aplicação que usam o *CWF* poderão compartilhar a mesma biblioteca. Isso irá reduzir o consumo de recursos computacionais (memória *RAM*, *HD* e Processamento), pois a biblioteca dinâmica será carregada uma única vez para N instâncias de aplicação

que usam o *CWF*. Esse fator também irá facilitar a manutenção e atualização das versões do *CWF*, uma vez que é possível atualizar a biblioteca dinâmica em um único local e todas as instâncias de aplicação serão atualizadas automaticamente. O *CWF* também fornece a possibilidade de utilizar compilação estática.

A classe `CppWebServer` é encapsulada pela `CppWebApplication` e derivada da `QTcpServer`, que é a classe do *Qt Framework* responsável por fornecer um servidor *TCP*. Portanto, a classe `CppWebServer` é responsável por atuar como um servidor *HTTP* e receber requisições. Para cada nova requisição, o método `incomingConnection` desta classe é chamado. Cada nova requisição é colocada dentro de um `QThreadPool`, que é uma classe responsável por gerenciar uma coleção de *threads*, que irá lê-la e respondê-la.

3.3.4 Controllers

Os *controllers* do *CWF* foram inspirados nos Java *Servlets*, que permitem manusear requisições e gerar conteúdos dinâmicos. Implementar *controllers* no *CWF* consiste em criar uma classe derivada de `Controller` e concretizar um ou mais métodos virtuais à sua escolha: `doGet`, `doPost`, `doPut`, `doDelete`, `doTrace` and `doOptions`. Cada método recebe dois parâmetros passados por referência: `Request`, que mantém informações sobre a requisição; e `Response`, que fornece mecanismos para responder a uma requisição.

O Algoritmo 15 mostra um *controller* que é capaz de lidar com duas requisições (*GET* e *POST*). Ao receber uma requisição via *GET*, o *controller* irá devolver uma página de *login*. Após o preenchimento dos dados da página *login*, o usuário poderá clicar em um botão para enviar uma requisição via *POST* com os dados de *login* e senha. Caso o *login* e a senha estejam corretos, a página inicial será retornada para o usuário; caso contrário, o usuário será redirecionado para a página de autenticação. As sessões armazenam informações sobre os clientes conectados ao servidor.

A única regra que os desenvolvedores devem cumprir para armazenar uma informação dentro de uma sessão é que a classe que contém a informação deve ser derivada de `QObject`. O método `fillQObject` é usado para configurar automaticamente os atributos de qualquer classe derivada de `QObject`, desde que os nomes do formulário *HTML* tenham exatamente o mesmo nome das propriedades da classe e a classe disponibilize os métodos

set na sessão `public slots`. O método `getSession` é utilizado para obter a sessão atual e o método `addAttribute` adiciona um objeto na sessão atual.

Algoritmo 15 Exemplo de uso de controllers e sessões

```

1: class LoginController : public CWF :: Controller
2: {
3: public :
4:   void doGet(CWF :: Request &req, CWF :: Response &resp) const override
5:   {
6:     req.getRequestDispatcher("/pages/login.view").forward(req, resp);
7:   }
8:   void doPost(CWF :: Request &req, CWF :: Response &resp) const override
9:   {
10:    QString login(req.getParameter("login"));
11:    QString pwd(req.getParameter("passwd"));
12:    if(login == "user" && pwd == "pwd")
13:    {
14:      Use usr;
15:      req.fillQObject(&usr);
16:      req.getSession().addAttribute("usr", usr);
17:      req.getRequestDispatcher("/pages/home.view").forward(req, resp);
18:    }
19:    else
20:    {
21:      req.getDispatcher("/pages/login.view").forward(req, resp);
22:    }
23: };

```

Todos os *controllers* devem ser registrados para que o servidor saiba que eles existem. Com isso, quando o servidor receber uma nova requisição, poderá encaminhá-la para o *controller* responsável por tratá-la. Uma possível desvantagem dessa abordagem é que as aplicações feitas em *CWF* podem se tornar menos flexíveis já que os *controllers* não podem ser modificados ou adicionados dinamicamente. Entretanto, é possível utilizar recursos de ligação dinâmica para contornar esse problema.

Em aplicações web, filtros são usados para interceptar requisições, antes que cheguem aos *controllers*, podendo conter uma validação prévia, como: verificar se a sessão do usuário está ativa e, em caso positivo, o direciona para o *controller*; caso contrário, pode redirecionar o usuário para a tela de autenticação. Isso elimina a necessidade de se criar métodos de validação para todos os *controllers*. Utilizar os filtros propostos pelo *CWF* é simples, pois basta criar uma classe derivada de `Filter` e redefinir o método `doFilter`. Os filtros também

precisam ser registrados no construtor da classe `CppWebApplication`. O Algoritmo 16 mostra um exemplo de filtro criado em uma aplicação *CWF*.

O exemplo apresentado no Algoritmo 16 mostra como se cria um filtro no *CWF*. Se a requisição envolver apenas um arquivo *CSS*, *PNG* ou a tela de login, o filtro despacha a requisição. Caso seja qualquer outro recurso do servidor, o filtro irá verificar se o usuário está autenticado e se a sua sessão não está expirada. Se o usuário não estiver autenticado ou a sessão estiver expirada, o filtro irá encaminhá-lo para a página de autenticação. Caso contrário, o usuário será dirigido ao recurso que solicitou. Isso evita que a validação seja feita em todos os *controllers* do sistema, pois, antes de passar pelo *controller*, a requisição irá passar pelo filtro.

Algoritmo 16 Exemplo de um Filtro em *CWF*

```

1: class SessionValidatorFilter : public CWF :: Filter
2: {
3: public :
4:     void doFilter(CWF :: Request &req,
5:                 CWF :: Response &resp, CWF : FilterChain &chain)
6:     {
7:         QString url(req.getRequestURL());
8:         if(url.endsWith(".css") || url.endsWith(".PNG"))
9:         {
10:            chain.doFilter(req, resp);
11:        }
12:        else if(url != "/login")
13:        {
14:            QObject *obj = req.getSession().getAttribute("usr");
15:            if(obj == nullptr || req.getSession().isExpired())
16:            {
17:                req.getRequestDispatcher("/pages/login.view").forward(req, resp);
18:            }
19:            else
20:            {
21:                chain.doFilter(req, resp);
22:            }
23:        }
24:        else
25:        {
26:            chain.doFilter(req, resp);
27:        }
28:    }
29: };

```

3.3.5 Páginas web e a CSTL

A *CSTL* (*C++ Server Pages Standard Tag Library*) é uma biblioteca de *tags* usada para gerar conteúdo dinâmico. Essa biblioteca de *tags* permite separar a camada de apresentação da camada de negócios e é utilizada para criar aplicações na arquitetura *MVC* (*Model-View-Controller*). O desenvolvedor pode criar classes para a camada de negócios (*Model*), usar a *CSTL* dentro das páginas web para apresentar dados (*View*) e usar os *controllers* como intermediários entre as duas camadas (*Controller*). Ela foi inspirada na biblioteca *JSTL* (*JavaServer Pages Standard Tag Library*) do Java. A *CSTL* permite utilizar objetos C++ dentro de páginas *view*, usando *tags* ao invés de código de programação C++. Até o momento, a *CSTL* possui as seguintes *tags*: *out*, *for*, *if* e *import*. Mesmo tendo apenas quatro *tags* até o momento, a combinação delas é flexível e poderosa o suficiente para construir vários tipos de aplicações *web*. Existem algumas regras para o uso da *CSTL*:

1. Todos os objetos devem ser derivados de *QObject*;
2. Todos os métodos usados pela *CSTL* devem estar na seção *public slots* para que possam ser manipulados via *reflection*;
3. Não é possível modificar os valores dos objetos – apenas consultá-los;
4. O valor retornado pelos métodos usados dentro da *CSTL* devem ser de um tipo primitivo da linguagem C++, ou *std::string*, ou *QString*.

A classe **Customer**, apresentada no Algoritmo 17, é um exemplo de classe que pode ser utilizada dentro das páginas *view*.

Algoritmo 17 Uma classe *Model* preparada para ser utilizada pela *CSTL*

```

1: class Customer : public QObject
2: {
3:     Q_OBJECT
4:     char gender; int id; QString name;
5:     public slots :
6:         char getGender() const{return gender;}
7:         int getId() const{return id;}
8:         QString getName() const{return name;}
9:         void setGender(char value) const{gender = value;}
10:        void setId(int value){id = value;}
11:        void setName(const QString &value){name = value;}
12: };

```

No Algoritmo 18, é mostrado um *controller* que passa uma lista de objetos para uma página *view*.

Algoritmo 18 Enviando objetos para a página via *Controller*

```

1: class CustomerController : public CWF :: Controller
2: {
3: public :
4:     void doGet(CWF :: Request &req, CWF :: Response &resp) override
5:     {
6:         ...
7:         CWF :: QListObjectqListObject;
8:         qListObject.add(&customerOne);
9:         qListObject.add(&customerTwo);
10:        qListObject.add(&customerThree);
11:        qListObject.add(&customerFour);
12:        req.getRequestDispatcher("/pages/customer.view").forward(req, resp);
13:    }
14: };

```

A página que recebe a lista de objetos passados pelo **CustomerController** é apresentada na Algoritmo 19. Ao utilizar a *CSTL*, os métodos são chamados usando `#{class.method}` como modelo e os valores são mostrados usando a tag `out`. A tag `for` pode ser usada para iterar sobre uma coleção de objetos e sobre valores numéricos (`<for var="i"from="1"to="10"increment="1">`). Neste exemplo, a tag `for` é usada para acessar cada objeto dentro da lista de clientes recebida de **CustomerController**. A tag `if` impõe uma condição: somente clientes masculinos (gênero == 'M') serão mostrados.

Algoritmo 19 Uso da *CSTL* *for*, *if* e *out* na *View*

```

1: < html >
2:   < body >
3:     < table >
4:       < tr > < td > < b > Name < /b > < /td > < /tr >
5:       < for items = "customersList" var = "customer" >
6:         < if var = "#{customer.getGender}" equal = "M" >
7:           < tr >
8:             < td > < out value = "#{customer.getId}" \ > < /td >
9:             < td > < out value = "#{customer.getName}" \ > < /td >
10:          < /tr >
11:        < /if >
12:      < /for >
13:    < /table >
14:  < /body >
15: / < html >

```

3.4 Acesso a Banco de Dados

Foi desenvolvida a classe `CWF::SqlDatabaseStorage` que permite o acesso aos bancos de dados: *IBM DB2*, *Borland InterBase Driver*, *MySQL Driver*, *Oracle Call Interface Driver*, *ODBC Driver*, *PostgreSQL*, *SQLite* e *Sybase Adaptive Server*. Essa classe encapsula a classe `QSqlDataBase` do *Qt Framework* e é reaproveitada pela mesma *thread* quantas vezes for necessário de modo a evitar a abertura e fechamento de conexões com o banco de dados a todo instante, uma vez que a classe `QSqlDataBase` só pode ser usada dentro do processo que a criou. Destaca-se que, devido ao *CWF* ser um projeto *open source* e estar sob licenciamento *MIT* no *GitHub*, ele vem recebendo muitas contribuições de desenvolvedores e cientistas de diversas partes do mundo. Dentre as contribuições recebidas, vale destacar que o módulo *ORM* foi desenvolvido pelo pesquisador Sylvain Meylan. O módulo ainda está em fase experimental, mas já possui algumas funcionalidades implementadas, como criar, modificar e atualizar uma tabela no banco de dados. O Algoritmo 20 mostra como preparar uma classe que será utilizada dentro do módulo *ORM* do *CWF*.

Algoritmo 20 Exemplo de uma classe *model* para ser utilizada no módulo *ORM*

```

1: #include < cwf/model.h >
2: class UserModel : public CWF :: Model
3: {
4:     Q_OBJECT
5:     Q_PROPERTY(QString name READ getName WRITE setName)
6:     Q_PROPERTY(QString phone READ getPhone WRITE setPhone)
7:     Q_PROPERTY(QString country READ getCountry WRITE setCountry)
8:     QString name, phone, country;
9: public :
10:     explicit UserModel(CWF :: SqlDatabaseStorage &connection) :
11:         CWF :: Model(connection, "usuario"){ }
12: public slots :
13:     QString getName() const{return name; }
14:     void setName(const QString &value){name = value; }
15:     QString getPhone() const{return phone; }
16:     void setPhone(const QString &value){phone = value; }
17:     QString getCountry() const{return country; }
18:     void setCountry(const QString &value){country = value; }
19: };

```

O Algoritmo 21 mostra como abrir uma conexão com o banco de dados, criar um *Controller* responsável por salvar os dados de um usuário no banco de dados e iniciar o servidor dentro da função *main*.

Algoritmo 21 Exemplo de uso do módulo *ORM*

```

1: #include < usermodel.h >
2: #include < cwf/cppwebapplication.h >
3: #include < cwf/sqldatabasestorage.h >
4: //Conexao com o banco de dados
5: CWF :: SqlDatabaseStorage conexao("QPSQL", "localhost", "postgres",
6:                                   "postgres", "1234", 5432);
7: //salvar um usuario no banco de dados e retornar a resposta
8: class ORMController : public CWF :: Controller
9: {
10: public :
11:     void doGet(CWF :: Request &request,
12:               CWF :: Response &response) const override
13:     {
14:         UserModel userconexao;
15:         user.setName("Herik Lima");
16:         user.setPhone("+ 55 11 9 99999 - 0000");
17:         user.setCountry("Brazil");
18:         response.write(QByteArray("< html >< body >")+
19:                       (user.save() ? "Saved" : "Error")+
20:                       "< /body >< /html >");
21:     }
22: };

```

3.5 Considerações finais

Optou-se por apoiar o desenvolvimento de aplicações *web* em C++ por questões relacionadas a desempenho, retrocompatibilidade, quantidade de desenvolvedores, atualização de versões e retrocompatibilidade. Somente o C++, em seu padrão *ISO*, não é o bastante para se desenvolver um *web framework* com as características listadas neste projeto, uma vez que não fornece suporte às áreas vitais, como comunicação *TCP/IP*. Após uma análise com critérios bem definidos, escolhemos o *Qt Framework* como *framework* de apoio à proposta apresentada neste trabalho. Depois da escolha da linguagem de programação e do *framework* de apoio, desenvolvemos uma solução que deveria resolver ou minimizar os problemas descritos no Capítulo 2. O *C++ Web Framework* foi projetado para consumir poucos recursos computacionais (memória *RAM* e processamento), apresentar um bom tempo de resposta e, na visão dos usuários, ser fácil de usar e apresentar uma boa manutenibilidade, uma vez que foi criada a *CSTL* para evitar que os desenvolvedores misturem código C++ com código *HTML*. Por se tratar de um protótipo, a *CSTL* não apresenta tantas funcionalidades como a *JSTL* do Java EE, por exemplo, e o módulo *ORM* também está em fase experimental. No capítulo a seguir, nós definimos vários tipos de validação para verificar se o protótipo desenvolvido resolve ou pelo menos minimiza os problemas descritos no Capítulo 2.

4 Avaliação

Avaliar o *web framework* proposto de forma completa e extensiva para provar que ele resolve ou mitiga os problemas relacionados ao desenvolvimento de aplicações *web* exige muito esforço e tempo, e isso não está no escopo e não cabe no tempo determinado para a realização deste projeto de pesquisa, visto que o projeto e a implementação do *CWF* ocupou a maior parte do tempo. Entretanto, neste capítulo, são apresentados diversos tipos de avaliações pontuais realizadas com o *CWF* para dar indícios de que o *web framework* é fácil de usar, atende às necessidades dos desenvolvedores que precisam desenvolver aplicações *web* robustas, e que também apresenta um bom desempenho em relação ao tempo de resposta e uso de recursos computacionais.

Na Seção 4.1, é avaliada a complexidade de instalação e configuração do *web framework* proposto em comparação com a principal solução do mercado para aplicações *web* de alto desempenho, levando em conta o número de itens necessários para se instalar e configurar em cada uma das soluções para criação de páginas estáticas e dinâmicas, acesso a banco de dados, *web services REST* e manipulação de arquivos *JSON* e *XML*. A Seção 4.2 trata da avaliação do nível de dificuldade de aprendizagem e uso do *web framework* proposto, na visão dos usuários, por meio de experimentos realizados em ambiente controlado. A Seção 4.3 mostra as avaliações de desempenho e a Seção 4.4 relata avaliações realizadas em ambientes de desenvolvimento real.

4.1 Avaliação da instalação e configuração inicial

Um dos objetivos do *CWF* é fornecer aos desenvolvedores uma alternativa simples para o desenvolvimento *web*. Desta forma, espera-se que os desenvolvedores não precisem despender um grande esforço para instalar e configurar o *web framework* de tal forma que ele fique pronto para o desenvolvimento de uma aplicação *web* típica.

Antes de avaliar o uso do *framework* com usuários, entretanto, decidiu-se comparar as tarefas necessárias para instalar e configurar o *CWF* com as tarefas necessárias para atingir o mesmo objetivo utilizando outro *web framework*. Para esta análise, foi selecionado o Java EE 8, que embora seja tratado como uma especificação, fornece um conjunto

de ferramentas muito similar ao *CWF* e é o mais utilizado em aplicações *web* de alto desempenho. Nesta avaliação, as seguintes métricas foram consideradas:

1. Quantidade de bibliotecas e/ou *frameworks* a serem instalados;
2. Quantidade de arquivos a serem configurados.

A aplicação *web* típica, utilizada como referência, deve ter acesso a banco de dados, manipulação de arquivos *XML* e *JSON* e uso da *CSTL/JSTL*. Nesta avaliação, foi considerado o ambiente Linux com a distribuição *Deepin 15.5*.

Para instalar o *CWF* e as bibliotecas que atendam às necessidades da aplicação *web* de referência foi necessário executar os seguintes passos:

1. Fazer o *download* e instalar as bibliotecas do *Qt Framework*;
2. Fazer o *download* e instalar a *IDE Qt Creator*;
3. Instalar o *driver* do banco de dados;
4. Instalar o *CWF*;
5. Criar uma aplicação *Qt Console*;
6. Inserir os módulos *network* e *xml* no arquivo de projetos;
7. Inserir o caminho para a biblioteca dinâmica do *CWF*;
8. Criar um arquivo de configuração chamado *CPPWeb.ini*.

Para instalar o Java EE 8 e as bibliotecas que atendam às necessidades descritas, foi necessário executar os seguintes passos:

1. Fazer o *download* e instalar o Java;
2. Fazer o *download* e instalar o *Tomcat*;
3. Fazer o *download* do *org.json*;
4. Fazer o *download* da *taglibs-standard-compat*;
5. Fazer o *download* da *taglibs-standard-impl*;
6. Fazer o *download* da *taglibs-standard-jstlel*;
7. Fazer o *download* da *taglibs-standard-spec*;
8. Fazer o *download* do *driver* do banco de dados;
9. Fazer o *download* e instalar o *IDE Eclipse* (ou outra IDE de desenvolvimento);
10. Criar uma aplicação *Java EE (Dinamic Web Application)*;
11. Configurar o *Tomcat* para a aplicação;

12. Adicionar o projeto ao *Tomcat*;
13. Adicionar a *API* de *servlets* no *buildpath* do projeto;
14. Adicionar o *driver* de banco de dados dentro das *libraries*;
15. Adicionar o *org.json* dentro das *libraries*;
16. Adicionar o *taglibs-standard-compat* dentro das *libraries*;
17. Adicionar o *taglibs-standard-impl* dentro das *libraries*;
18. Adicionar o *taglibs-standard-jstlel* dentro das *libraries*;
19. Adicionar o *taglibs-standard-spec* dentro das *libraries*;
20. Criar um arquivo de configuração chamado *web.xml*.

O número de passos necessários para criar uma aplicação *web*, como a usada como referência, é visivelmente maior quando o *Java EE* é utilizado. Isso se deve ao fato de o *Qt* já trazer consigo muitos dos recursos que os desenvolvedores irão necessitar para criar as suas aplicações. Em Java, é necessário instalar muitas bibliotecas auxiliares para obter uma série de funcionalidades não disponíveis nativamente pela linguagem, ou por sua especificação *Enterprise Edition*. O grande problema começa quando uma biblioteca possui dependência de outra, que depende de outra, e assim sucessivamente, gerando assim o *dependency hell*, situação em que é necessário instalar diversas bibliotecas para implementar uma única funcionalidade, e mantê-las atualizadas e resolver os conflitos entre elas. Esse é um problema tão grande, e ao mesmo tempo tão comum, que existem inúmeras ferramentas de gerenciamento de bibliotecas, como o *Maven* para o Java, por exemplo.

O problema de utilizar ferramentas de gerenciamento de dependências é que, para utilizar o *Maven* ou outra ferramenta similar é necessário há necessidade de aprender como funciona, suas regras de sintaxe e etc. Infelizmente, esse processo não é tão simples, e até mesmo programadores experientes têm dificuldades com este processo. Uma pesquisa em fóruns de desenvolvimento irá revelar inúmeros desenvolvedores relatando problemas para configurar essas ferramentas. Isso sem contar que existem inúmeros cursos diferentes apenas para ensinar os desenvolvedores a manusearem essas ferramentas. Evidentemente que o *CWF* não resolve o problema de forma definitiva, mas por utilizar o *Qt*, e pelo fato de o *Qt* possuir uma ampla gama de áreas de atuação, o *dependency hell* pode ser eliminado para alguns usuários e minimizado para outros.

É importante mencionar que o *CWF* não elimina a flexibilidade, uma vez que os desenvolvedores podem utilizar bibliotecas externas ao Qt, se assim desejarem. Entretanto, utilizar bibliotecas externas exige o gerenciamento e manutenção das dependências e dos conflitos que surgem neste contexto.

4.2 Avaliação da facilidade de aprendizagem e uso

Conforme mencionado anteriormente, o *CWF* foi construído para ser um *web framework* fácil de ser instalado, configurado e utilizado. Além de avaliar os passos necessários para sua instalação e configuração para construir uma aplicação *web* típica, uma avaliação foi feita com usuários. Para isso, três etapas distintas foram realizadas. A primeira refere-se à avaliação da capacidade de instalação e criação de uma página estática utilizando o *web framework*, e é descrita na Seção 4.2.1. A segunda refere-se à capacidade de usuários de instalar, configurar e criar aplicações dinâmicas simples, e está descrita na Seção 4.2.2. A terceira refere-se à avaliação de uma nova versão do *web framework* após mudanças arquiteturais realizadas com base nos resultados obtidos nas duas primeiras etapas. Nesta terceira etapa, os participantes também avaliaram a criação e uso de serviços *web* e a criação e uso de banco de dados. Esta terceira etapa é descrita na Seção 4.2.3.

4.2.1 Primeira etapa: instalação e criação de página estática

A primeira avaliação relacionada à facilidade de aprendizagem e uso do *CWF* teve como objetivo verificar se desenvolvedores seriam capazes de instalar o *web framework* e colocar uma aplicação *web* com uma página estática no ar, e medir o tempo que levaram para realizar esta tarefa. Para este estudo inicial, foram enviados convites para quatro participantes. As seguintes tarefas foram escolhidas para que os participantes pudessem desenvolver: i) fazer o *download* do *CWF*; ii) instalá-lo em sua máquina; iii) criar uma aplicação *web* simples, com uma página estática; e iv) colocar a aplicação no ar. Após executar essas tarefas, os participantes preencheram um questionário online cujo objetivo é identificar o perfil do desenvolvedor e obter sua opinião sobre o processo de utilizar o *CWF* para criar e colocar no ar uma aplicação simples.

Os quatro participantes possuíam nível de escolaridade semelhante: todos tinham uma graduação acadêmica na área de tecnologia, e dois já haviam feito uma especialização, como mostra a Figura 6. Três dos quatro participantes possuíam bastante experiência no setor de T.I., tendo mais de cinco anos de atuação na área, enquanto apenas um possuía menos de três anos de experiência, conforme mostra a Figura 7. Metade dos participantes possuíam experiência de mais de 5 anos no desenvolvimento de aplicações *web*, enquanto metade possuía menos do que três anos e experiência (ver Figura 8).

Figura 6 – Escolaridade dos participantes

Fonte: Herik Lima de Castro

Figura 7 – Experiência no setor de T.I.

Fonte: Herik Lima de Castro

Figura 8 – Experiência com desenvolvimento web

Fonte: Herik Lima de Castro

Após receber instruções simples sobre como fazer o download, instalar, configurar e desenvolver uma aplicação *web* usando o *CWF*, os desenvolvedores executaram as tarefas designadas a eles e emitiram uma opinião sobre o grau de dificuldade para realizar as tarefas. A Figura 9 indica o nível de dificuldade apontado pelos usuários para utilizar o *CWF* e colocar uma aplicação simples no ar. Observa-se que, três participantes disseram que o *web framework* é fácil ou muito fácil de usar. Esses resultados foram importantes para entender se era necessário fazer alguma modificação nos arquivos de configuração e na estrutura do *web framework*. Em princípio, os resultados foram considerados satisfatórios e nenhuma alteração foi realizada.

Figura 9 – Nível de dificuldade para se configurar e instalar o *CWF*

Fonte: Herik Lima de Castro

Dos dois desenvolvedores com mais de 5 anos de atuação na área, um considerou o nível de dificuldade para se configurar e instalar o *CWF* como fácil e o outro como muito fácil.

4.2.2 Segunda etapa: instalação, configuração e criação de página dinâmica

A segunda avaliação teve o objetivo de avaliar se desenvolvedores conseguiam instalar, configurar e criar uma página dinâmica usando o *CWF*. Além disso, os tempos para completar as tarefas sugeridas para os desenvolvedores foram medidas e a opinião dos desenvolvedores sobre o grau de dificuldade foi coletada. Ao todo, 22 voluntários participaram desta parte da avaliação. No entanto, as respostas de 2 deles foram descartadas, um deles teve problemas técnicos com o computador, impedindo que o mesmo realizasse as atividades e o outro chegou com cerca de 02 horas de atraso, deixando de assistir às 02 horas de aula.

Planejamento

Este experimento foi planejado da seguinte forma:

- Sujeitos: desenvolvedores iniciantes e experientes na linguagem C++, com ou sem conhecimento prévio em desenvolvimento Web.
- Contato com os sujeitos: convite enviado por mensagem eletrônica para alunos, ex-alunos e profissionais de T.I cadastrados na base de e-mails da AGIT Informática Ltda., uma empresa de treinamento e de desenvolvimento de software.
- Local: laboratórios da AGIT Informática Ltda.
- Execução do experimento: o experimento terá quatro horas de duração, sendo duas horas para treinamento sobre desenvolvimento *web* utilizando o CWF e duas horas para a resolução de quatro (4) exercícios propostos, que serão detalhadas a seguir.
- Coleta de dados: ao final do experimento, cada participante deverá responder um formulário online contendo perguntas sobre sua formação, tempo de experiência no setor de T.I, tempo de experiência com desenvolvimento web e C++, nível de dificuldade para configurar e instalar o CWF, tempo que levou para resolver cada um dos exercícios e, se não conseguiu resolver algum exercício, deve explicar o motivo.
- Análise dos dados: utilização de estatísticas básicas descritivas.

Após um treinamento de duas horas, os participantes deverão fazer o *download*, instalar, configurar e então resolver quatro (4) exercícios propostos usando os mecanismos de desenvolvimento web do CWF e, ao final, deverão ter uma aplicação MVC. Cada exercício proposto está detalhado a seguir.

1. Criar um *Controller*, registrá-lo dentro do servidor *web* e programar o método *GET*, de modo que o método retorne uma página completa em *HTML* contendo um campo para que o usuário digite um número e um botão "enviar" para submeter todo o formulário via *POST* para o servidor. Essa atividade exige que toda a página seja construída dentro de um *Controller* e não é permitido executar chamada a arquivos externos.
2. Criar uma página *HTML*, mover todo o código *HTML* do *Controller* para a página e modificar o método *GET* do *Controller*, de modo que este método passe a chamar a página *HTML*. No entanto, para realizar a segunda atividade, os participantes devem usar métodos disponibilizados pelo *CWF* para ler a página e retorná-la ao navegador.
3. Programar o método *POST* do *Controller*, ler o parâmetro "número" enviado pelo navegador junto ao formulário, calcular o fatorial do número, gerar uma página dinâmica em *HTML* dentro do *Controller* contendo o resultado e enviá-la para o navegador. Essa atividade exige que a página dinâmica seja construída dentro do *Controller*, sendo vedado o uso de arquivos auxiliares.
4. Criar uma *View*, mover todo o código *HTML* do método *POST* para dentro da *View* e utilizar a *CSTL* para exibir o resultado. Nessa atividade, o participante deve utilizar um *Model*, preenchê-lo com o resultado, passar o objeto *Model* para a *View* utilizando o *Controller* como um intermediário e, em seguida, renderizar a página com o resultado para o navegador.

Ameaças à validade do estudo

Experimentos realizados com grupos de usuários podem ser questionáveis, pois, cada aluno ou grupo de alunos tem seu próprio perfil de atuação e experiência de desenvolvimento. Além disso, a didática utilizada pelo instrutor durante as aulas pode impactar diretamente no aprendizado e nos resultados. Existem chances de os resultados serem diferentes se esse mesmo experimento for aplicado com outro grupo de usuários e/ou com outro instrutor. No entanto, o que se pode fazer para minimizar esse risco é a realização de mais experimentos com mais participantes, pois, à medida em que se aumenta o número de participantes, se aumenta a confiabilidade nos resultados. A opinião de participantes inexperientes com

desenvolvimento *web* será dada sem referência anterior de comparação, diferentemente dos desenvolvedores experientes que emitirão opinião sobre o *CWF*, considerando experiência de uso de outros *frameworks*. Entretanto, considerando que o desenvolvimento *web* tende a ser complexo, acredita-se que mesmo a opinião de desenvolvedores sem experiência *web* pode ser um indício de que o *framework* é, de fato, fácil ou difícil de instalar e utilizar no desenvolvimento. Ainda, desenvolvedores sem experiência não possuem conhecimentos prévios de desenvolvimento *web* que podem facilitar a tarefa de usar o *CWF*.

Execução

Os convites para participação do experimento foram enviados para os alunos, ex-alunos e profissionais de T.I cadastrados na base de *e-mails* da AGIT Informática Ltda., empresa onde trabalha o autor desta dissertação. Os voluntários para participar do experimento foram divididos em três turmas diferentes, conforme a disponibilidade de cada um. Portanto, o experimento foi realizado em três momentos diferentes (11 de março de 2017, 27 de maio de 2017 e 20 de janeiro de 2018), mas todos receberam o mesmo treinamento e exercícios para resolver. Dessa forma, os resultados deste experimento são analisados em conjunto ao invés de serem feitas três análises independentes.

No dia do experimento, os participantes receberam um treinamento de duas (02) horas sobre desenvolvimento web usando o *CWF*. O treinamento teve uma parte teórica e uma parte prática. Na parte teórica, foram abordados tópicos fundamentais para o desenvolvimento de uma aplicação web, como: aplicações *web*, servidores *web*, navegadores, *servlets/controllers*, protocolo *HTTP*, requisição, resposta, páginas estáticas, páginas dinâmicas e uma apresentação sobre o *CWF*. Na parte prática, os participantes aprenderam a instalar e desenvolver aplicações utilizando Ubuntu 14.04, *Qt Creator* 5.5 e a versão mais atual do *CWF*. Alguns exemplos de implementação foram apresentados sobre como utilizar *Controllers*, preparar *Models* e a gerar páginas estáticas e dinâmicas. O material utilizado no treinamento pode ser visto neste link: https://docs.google.com/presentation/d/1DhszRoWP8ac1DTd0_yQqKUQcNwostaOD5t5awSoFNPI/edit?usp=sharing

Em seguida, os participantes tiveram duas horas para resolver os quatro exercícios propostos que já explicados anteriormente. Durante a resolução dos exercícios, dois participantes tiveram problemas técnicos e por isso os dados que geraram foram excluídos da análise. Um dos participantes enfrentou problemas técnicos com o computador que

estava utilizando e por não haver outro para a reposição, ele não conseguiu fazer nenhum dos exercícios. Outro participante chegou atrasado e não assistiu ao treinamento. Ao final do experimento, os participantes responderam ao questionário *online* criado no Google Formulários, mesmo aqueles cujos dados foram excluídos posteriormente da análise.

Resultados

Nesta seção, são apresentados os dados que foram coletados dos usuários. A Figura 10 mostra o nível de escolaridade dos participantes. Nota-se que todos os participantes possuem educação formal em tecnologia, e quase metade possui pós-graduação *lato* ou *stricto sensu*. Além disso, a maioria dos participantes atuam há mais de cinco anos na área de T.I., e cerca de 20% deles de um a três anos. Poucos são os desenvolvedores inexperientes que participaram da atividade, conforme mostra a Figura 11. Em contrapartida, a Figura 12 mostra que 50% dos participantes não tem experiência com desenvolvimento *web*, e boa parte (25%) possui pouca experiência – de um a três anos. Apenas 20% dos participantes tem experiência de mais de três anos com o desenvolvimento *web*. No que se refere à experiência com o desenvolvimento C++, os desenvolvedores possuem perfis diversificados. Destaca-se que cerca de 30% dos participantes tinham pouca ou nenhuma experiência, e cerca de 35% tinham mais de três anos de experiência.

Figura 10 – Escolaridade dos participantes

Fonte: Herik Lima de Castro

Figura 11 – Experiência no setor de T.I (Desenvolvimento de Sistemas)

Fonte: Herik Lima de Castro

Figura 12 – Experiência com desenvolvimento *web*

Fonte: Herik Lima de Castro

Figura 13 – Experiência com desenvolvimento em C++

Fonte: Herik Lima de Castro

A opinião dos participantes sobre o grau de dificuldade de instalar e configurar o *CWF* foi satisfatória, pois 80% dos participantes disseram que foi fácil ou muito fácil executar esta tarefa, conforme mostra a Figura 14. Todos os participantes com mais de um ano de experiência em desenvolvimento *web* consideraram o *CWF* fácil de instalar e configurar. Dentre os desenvolvedores com menos de um ano, ou sem experiência no desenvolvimento *web*, as opiniões ficaram bem distribuídas entre os níveis **muito fácil**, **fácil** ou **médio**.

Figura 14 – Nível de dificuldade para se configurar e instalar o *CWF*

Fonte: Herik Lima de Castro

Para os exercícios propostos, ao invés de perguntar o grau de dificuldade, optamos por registrar o tempo que os usuários levaram para executar a tarefa. Nota-se, ao observar a Figura 15, que a maioria dos participantes (65%) levou menos do que 20 minutos para fazer o exercício. Poucos participantes levaram mais do que 30 minutos. Para esse primeiro exercício, era esperado que os usuários levassem entre 20 a 30 minutos, uma vez que além do exercício, eles também teriam que criar e configurar um projeto em *CWF* do zero.

Figura 15 – Tempo gasto para resolver o primeiro exercício

Fonte: Herik Lima de Castro

O segundo exercício era mais complexo, uma vez que era necessário a criação do método *doPost*, implementação da função fatorial e criação de uma página dinâmica em *HTML* como resposta. Mas, novamente, a maioria dos participantes (70%) levou menos do que 20 minutos para realizar a tarefa, como mostra a Figura 16. Para essa atividade, era esperado que os participantes demorassem entre 20 a 30 minutos para realizá-la. Poucos demoraram mais do que 30 minutos.

Figura 16 – Tempo gasto para resolver o segundo exercício

Fonte: Herik Lima de Castro

O terceiro exercício era o mais simples, pois pedia que o usuário criasse uma nova página *HTML* e removesse o código *HTML* do método *doGet* para essa página, de modo a separar a camada de apresentação da camada de negócios. A Figura 17 mostra o tempo médio gasto pelos desenvolvedores na execução da tarefa. Note que, desta vez, a grande maioria (85%) levou menos do que 20 minutos na tarefa, sendo que mais da metade dos participantes levou menos do que 10 minutos. Para essa atividade era esperado que os participantes demorassem entre 10 e 20 minutos para realizá-la. Cerca de 30% dos participantes levaram mais do que 30 minutos, de fato, mas nenhum deles excedeu duas horas de atividade.

Figura 17 – Tempo gasto para resolver o terceiro exercício

Fonte: Herik Lima de Castro

Por fim, a Figura 18 indica o tempo gasto pelos usuários para solucionar o exercício 4, o mais complexo de todos, por requerer o uso de exibição de conteúdo dinâmico utilizando a *CSTL*. Naturalmente, o tempo médio gasto pelos participantes foi maior, mas não em demasia. Cerca de 40% dos participantes levaram menos do que 20 minutos para implementar a tarefa, 20% levou de 20 a 30 minutos e 30% levou mais do que 30

minutos. Por volta de 10% de participantes não conseguiram resolver a tarefa. Para essa atividade, era esperado que a maioria dos usuários demorassem entre 20 a 30 minutos e alguns demorassem até um pouco mais do que 30 minutos, pela sua complexidade, mas não poderiam exceder o tempo total de duas horas.

Figura 18 – Tempo gasto para resolver o quarto exercício

Fonte: Herik Lima de Castro

Os resultados desses estudos com desenvolvedores foram satisfatórios, uma vez que muitos deles tinham pouca ou nenhuma experiência com desenvolvimento *web* e/ou com C++, e, ainda assim, a maioria dos participantes achou o *framework* fácil de instalar e configurar, e conseguiu resolver os exercícios em tempo satisfatório (menor ou igual a 30 minutos) para quem nunca havia trabalhado com o *web framework CWF*. Além disso, os participantes conseguiram resolver todos os exercícios, com exceção de 10% de participantes que não conseguiu resolver o último exercício.

A quantidade de participantes e de dados coletados não permite uma análise significativa da correlação entre a experiência com programação e desenvolvimento *web* e o tempo de resolução dos exercícios. Entretanto, uma análise foi realizada para investigar se a experiência com desenvolvimento *web* exerceu alguma influência nos resultados obtidos. A Tabela 1 mostra a distribuição do tempo gasto em cada exercício por nível de experiência com desenvolvimento *web*. Note que a maioria dos participantes, considerando todos os níveis de experiência, levou menos do que 20 minutos para resolver os exercícios, e poucos levaram mais do que 30 minutos, com exceção do único participante com menos de um ano de experiência que levou mais do que 30 minutos em todos os exercícios. Este participante, em particular, teve problemas com a utilização do *Qt Framework* e por isso levou bastante tempo para fazer os exercícios. Dos 10 participantes sem nenhuma experiência, dois não conseguiram resolver o quarto exercício (ex4).

Tabela 1 – Distribuição do tempo gasto em cada exercício por nível de experiência com desenvolvimento *web*

Experiência		ex1	ex2	ex3	ex4	soma	% amostras
Nunca desenvolveu	<10	1	1	8	3	13	32.50%
	10-20	5	7	1	0	13	32.50%
	20-30	4	2	1	3	10	25.00%
	>30	0	0	0	2	2	5.00%
Menos de 1 ano	<10	0	0	0	0	0	0.00%
	10-20	0	0	0	0	0	0.00%
	20-30	0	0	0	0	0	0.00%
	>30	1	1	1	1	4	100.00%
Entre 1 e 3 anos	<10	2	0	1	0	3	15.00%
	10-20	2	3	3	3	11	55.00%
	20-30	1	1	1	1	4	20.00%
	>30	0	1	0	1	2	10.00%
Entre 3 e 5 anos	<10	0	1	1	0	2	25.00%
	10-20	2	1	1	0	4	50.00%
	20-30	0	0	0	0	0	0.00%
	>30	0	0	0	2	2	25.00%
Mais do que 5 anos	<10	0	1	0	0	1	12.50%
	10-20	1	0	2	2	5	62.50%
	20-30	0	0	0	0	0	0.00%
	>30	1	1	0	0	2	25.00%

Observou-se que, durante os testes, alguns participantes tiveram dificuldades com relação ao arquivo de configurações do *CWF* e também dúvidas sobre conversões entre tipos no *Qt*. O retorno dos participantes ao final da sessão sobre as principais dificuldades encontradas foi útil para definir estratégias para tornar o *CWF* ainda mais simples de instalar, configurar e usar.

Insights

Estes primeiros experimentos foram bastante importantes, pois, por meio deles, foi possível notar que os participantes escreviam muito para se criar os *servlets* e, em alguns casos, se perdiam nos nomes, o que nos levou a modificar o nome *HttpServlet* para *Controller* e os nomes *HttpServletRequest* e *HttpServletResponse* para *Request* e *Response*, respectivamente. Um outro recurso interessante que foi adicionado após os experimentos foi o *CWF::Variant*, permitindo que o usuário passe objetos primitivos, como *int*, *double* e *string* diretamente para a *view*, sem a necessidade de se criar um objeto derivado de *QObject* apenas para isso. Outra mudança significativa foi na extensão das *views*, que

tiveram sua extensão modificada de *.xhtml* para *.view*, tornando a arquitetura *MVC* mais visível aos usuários.

4.2.3 Terceira etapa: criação de serviços web e banco de dados

A terceira avaliação foi um experimento controlado realizado com 11 voluntários e seguiu exatamente o mesmo planejamento do experimento descrito na Seção 4.2.2. A diferença deste experimento é que um exercício extra foi criado para o desenvolvimento de *web services* e acesso a banco de dados. Além dos quatro exercícios iniciais, os usuários deverão realizar o quinto exercício em ambiente não controlado e enviar as respostas do exercício utilizando um repositório de código.

Desta forma, ao término do tempo para realização dos exercícios em sala, os voluntários receberão uma aula com duração de uma hora mostrando a criação de *web services REST* com acesso a banco de dados PostgreSQL. O exemplo será constituído de um *web service REST* capaz de consultar, alterar, deletar e inserir usuários e telefones de uma agenda de contatos dentro de uma tabela no banco de dados PostgreSQL, por meio dos métodos *GET*, *PUT*, *POST* e *DELETE*, utilizando o formato *JSON* para a transmissão e respostas. Ao final, será fornecido o *script* de uma tabela *SQL* chamada “logradouro”, que irá conter os campos: id, país, estado, cidade, bairro, logradouro e cep. Todos os participantes deverão instalar o banco de dados PostgreSQL, criar a tabela “logradouro” com o *script* fornecido e criar um *web service REST* que permita a inserção, remoção, alteração e exclusão de endereços utilizando os métodos *GET*, *POST*, *PUT* e *DELETE* do *web service*. Após a realização dos exercícios, os mesmos deverão ser colocados no *GitHub* e o link para o repositório deverá ser fornecido no preenchimento do formulário, de modo que seja possível verificar se os exercícios foram feitos corretamente.

Ameaças à validade do estudo

As ameaças deste estudo são as mesmas do estudo apresentado na Seção 4.2.2, com o acréscimo das limitações de experimentos em ambiente não controlado. Como os participantes realizarão a atividade em ambiente não controlado, não é possível medir o tempo de conclusão dos exercícios com precisão.

É possível que os participantes não forneçam os valores medidos corretamente. Um outro fator de ameaça é o possível atraso ou a não conclusão da tarefa por parte dos desenvolvedores.

Execução

Os convites para participação do experimento foram enviados para os alunos, ex-alunos e profissionais de T.I cadastrados na base de e-mail da AGIT Informática Ltda. O experimento foi realizado em 09 de março de 2019 e todos receberam o mesmo treinamento e exercícios para resolver.

No dia do experimento, os participantes receberam um treinamento de duas (02) horas sobre desenvolvimento web usando o *CWF*. O treinamento teve uma parte teórica e uma parte prática. Na parte teórica, foram abordados tópicos fundamentais para o desenvolvimento de uma aplicação *web*, tais como: aplicações *web*, servidores *web*, navegadores, *servlets/controllers*, protocolo *HTTP*, requisição, resposta, páginas estáticas, páginas dinâmicas e uma apresentação sobre o *CWF*. Na parte prática, os participantes aprenderam a instalar e desenvolver aplicações utilizando *Deepin 15*, *Qt Creator 5.5* e a versão mais atual do *CWF*. Alguns exemplos de implementação foram apresentados sobre como utilizar *Controllers*, preparar *Models* e a gerar páginas estáticas e dinâmicas.

Em seguida, os participantes tiveram duas horas para resolver os quatro exercícios propostos e exemplificados na Seção 4.2.2. Na última hora de aula, foi mostrado aos participantes a criação de *web services REST* e o acesso ao banco de dados *PostgreSQL*. Após a apresentação, foi fornecido um *script SQL* para a criação de uma tabela no banco de dados e quatro exercícios relacionados a *web services* para que os participantes resolvessem em casa em até 2 semanas. Ao final do experimento, todos os participantes responderam ao questionário online criado no Google Formulários.

Resultados

Os participantes deste experimento possuem diferentes níveis de escolaridade, conforme mostra a Figura 19. A maioria possui graduação e poucos possuem uma especialização. Pouco menos de um terço não possui educação superior formal. Grande parte dos participantes tem experiência de mais de três anos na área de TI, e poucos são os que não

atuam na área (ver Figura 20). Neste grupo, mais de 60% dos participantes nunca havia desenvolvido para a *web* ou tem menos de um ano de experiência. Poucos são os que têm três ou mais anos de experiência, conforme mostra a Figura 21. Além disso, muitos nunca haviam programado em C++ e poucos possuem experiência de mais de três anos com a linguagem.

Figura 19 – Escolaridade dos participantes

Fonte: Herik Lima de Castro

Figura 20 – Experiência no setor de T.I

Fonte: Herik Lima de Castro

Figura 21 – Experiência com desenvolvimento web

Fonte: Herik Lima de Castro

Figura 22 – Experiência com desenvolvimento em C++

Fonte: Herik Lima de Castro

Semelhantemente aos outros experimentos, a grande maioria (80%) achou fácil ou muito fácil instalar e configurar o *CWF*, conforme mostra a Figura 23. Os tempos necessários para desenvolver os exercícios 1 a 4 foram semelhantes aos tempos dos experimentos anteriores, incluindo a porcentagem de participantes que não conseguiu resolver o quarto exercício. As Figuras 24, 25, 26 e 27 indicam o tempo gasto pelos usuários para solucionar cada exercício.

Figura 23 – Nível de dificuldade para se configurar e instalar o *CWF*

Fonte: Herik Lima de Castro

Figura 24 – Tempo gasto para resolver o primeiro exercício

Fonte: Herik Lima de Castro

Figura 25 – Tempo gasto para resolver o segundo exercício

Fonte: Herik Lima de Castro

Figura 26 – Tempo gasto para resolver o terceiro exercício

Fonte: Herik Lima de Castro

Figura 27 – Tempo gasto para resolver o quarto exercício

Fonte: Herik Lima de Castro

Os resultados desta primeira etapa foram satisfatórios e obtiveram resultados parecidos com os experimentos realizados anteriormente. No entanto, ao analisar os dados do exercícios que foram propostos para casa, os resultados não foram tão satisfatórios. Todos os participantes atrasaram as entregas não em razão da complexidade dos exercícios, mas da falta de tempo para fazer os exercícios em razão de problemas de ordens pessoais e profissionais. Então os prazos para a entrega foram então estendidos em algumas semanas e, após mais atrasos, os participantes responderam os questionários sem sequer tentar solucionar os problemas propostos. Neste sentido, percebeu-se que o experimento não controlado, de fato, oferece muitos riscos, pois os participantes não se comprometem com as atividades como quando estão em ambiente controlado.

Os resultados, portanto, possuem grande influência dos problemas comentados anteriormente, mas, ainda assim, estão apresentados aqui para discussão. As Figuras 28, 29, 30 e 31 mostram se os participantes conseguiram resolver os exercícios de *GET*, *POST*, *PUT*

e *DELETE*, respectivamente. Note que quase metade dos participantes não conseguiram resolver os exercícios e os que resolveram não cronometraram o tempo gasto com a tarefa.

Figura 28 – Conseguiu solucionar o exercício com *GET*

Fonte: Herik Lima de Castro

Figura 29 – Conseguiu solucionar o exercício com *POST*

Fonte: Herik Lima de Castro

Figura 30 – Conseguiu solucionar o exercício com *PUT*

Fonte: Herik Lima de Castro

Figura 31 – Conseguiu solucionar o exercício com *DELETE*

Fonte: Herik Lima de Castro

Apesar dos resultados relativos ao exercício 5 serem insatisfatórios como um todo, principalmente por conter dados incompletos, os resultados dos exercícios de 1 a 4 foram satisfatórios. Além disso, alguns participantes conseguiram resolver o exercícios 5. Os que não conseguiram resolver foi em razão de não tentar resolver por causa do tempo, problemas pessoais ou por dificuldades com a tarefa.

4.3 Avaliação de desempenho

Aumentar o desempenho das aplicações *web* tanto em termos de reduzir o tempo de resposta, quanto do consumo de recursos computacionais, é um dos objetivos deste projeto de pesquisa com a criação do *CWF*. Desta forma, uma avaliação de desempenho foi realizada para verificar se as aplicações desenvolvidas com o *CWF* conseguem ser mais eficientes do que aplicações desenvolvidas em outras linguagens. Para esta avaliação, foi escolhido comparar aplicações *CWF* com aplicações escritas em Java EE, uma vez que esta é uma das ferramentas mais populares e utilizada para desenvolver aplicações que exigem alto desempenho (MILLARES, 2015).

Os três primeiros cenários de testes consistiram em criar *web services REST* para realizar cálculos matemáticos (fatorial, números primos e fibonacci). Os *web services* deveriam receber uma requisição via método *HTTP - GET*, ler um parâmetro chamado “number” enviado pela requisição e realizar o cálculo com base no parâmetro recebido. Após o cálculo, o *web service* deveria serializar um objeto *JSON* contendo um campo chamado “result”, inserir o resultado do calculo dentro do campo “result” do *JSON*, configurar o “Content-Type” da requisição para “application/json” e retornar a resposta para o servidor.

O quarto e último teste consistia em criar um *web service* para gerar uma página *HTML* dinâmica, contendo em seu corpo 10 mil caracteres. Cada teste consistiu em efetuar 10 mil requisições para o servidor e medir o tempo de resposta, o consumo de memória *RAM* e o uso da unidade processamento do servidor. Todos os testes foram replicados três vezes e os resultados aqui reportados são a média dos valores obtidos.

O desempenho do *CWF* foi comparado com o desempenho do Java 8 usando *Tomcat* 8. A escolha do Java para a comparação se deve ao fato de que a maior parte dos grandes sites o utilizam em seu desenvolvimento (MILLARES, 2015) e a escolha do *Tomcat* se deve ao fato deste ser o servidor *web* mais popular no mundo Java (SALNIKOV-TARNOVSKI, 2017). Os testes foram realizados em uma configuração cliente-servidor utilizando dois computadores. O servidor foi um Notebook Samsung RV 411, Ubuntu 17.10 (64 bits), SSD Kingston 120 GB, 8 GB de RAM e processador Intel Core i5 M 480 2.67 GHz. No servidor, foi utilizado o Monitor de Processos do sistema operacional, GCC 5.3.1, Jemalloc, Qt 5.10.1, *CWF* Commit 174, Java 1.8.0, *Tomcat* 8.0.51 e os códigos foram otimizados para o *Tomcat*. No C++, foi utilizada o flag de otimização -O3. O cliente foi um Notebook Samsung RV 420, Deepin 15.4.1 (64 bits), SSD Kingston 120 GB, 8 GB de RAM e processador Intel (R) Celeron (R) CPU B800 1.50 GHz. No cliente, foi utilizado o *Weighttp* Commit 29 para efetuar os testes de carga. O roteador utilizado foi um Archer C7 Router AC 1750 - TP-LINK e as máquinas foram conectadas via cabo de rede.

Ambos os servidores *web* (Java e *CWF*) tiveram seus logs desligados para que não impactassem no desempenho. Após o término de cada teste, as duas máquinas eram reiniciadas para garantir uma condição de igualdade para ambos os *web frameworks*. Durante os testes, foram medidos os picos de consumo inicial e final de memória *RAM* e processador, e também o tempo de resposta para as requisições. Os quatro testes aplicados se basearam em testes que são normalmente aplicados em comparações de *web frameworks* (TECHEMPOWER, 2018).

Ameaças

Embora todos estes testes de desempenho sejam questionáveis e passíveis de críticas, pois está sendo feita uma comparação entre um *web framework* e a especificação Java EE, o Java EE fornece ferramentas similares ao *CWF*. O teste realizado permite a avaliação do *web framework* em relação a um valor objetivo que pode ser considerado de forma objetiva ou em comparação com um outro *web framework*. Por ser *open source*, e estar sob licenciamento *MIT*, o *CWF* já recebeu inúmeras contribuições de pessoas de diversos países do mundo, o que facilita e ajuda na descoberta de falhas de segurança, *bugs* e criação de novos recursos e módulos.

Execução

O primeiro teste consistiu em calcular o fatorial do número 10. A Tabela 2 mostra os resultados obtidos em cada teste. Em média, o *CWF* teve um consumo inicial de memória *RAM* de 133,26 vezes menor do que o Java e o consumo inicial de processamento de ambos ficou empatado. Durante as requisições, o consumo de memória *RAM* do *CWF* foi 121,97 vezes menor do que o Java, o pico de consumo de processamento do *CWF* foi 3,96 vezes menor do que o Java e o *CWF* conseguiu responder às requisições 4,56 vezes mais rápido, como mostra a Figura 32.

Tabela 2 – Fatorial (10) - Resultados

	Teste 1		Teste 2		Teste 3	
	CWF	Java 8	CWF	Java 8	CWF	Java 8
RAM inicial (KB)	792,00	105.779,20	796,00	105.062,40	792,00	104.038,40
RAM final (KB)	1.228,80	154.828,80	1.228,80	133.632,00	1.228,80	147.968,00
Processador % (Ini)	0,00	0,00	0,00	33,00	0,00	28,00
Processador % (Fim)	17,00	94,00	17,00	91,00	17,00	81,00
Tempo resposta (ms)	628,00	3.019,00	667,00	3.011,00	641,00	3.865,00

Fonte: Herik Lima de Castro

O segundo teste consistia em verificar se o número 997 era primo ou não. A Tabela 3 mostra os resultados obtidos em cada teste. Em média, o *CWF* teve um consumo inicial de memória *RAM* 132,30 vezes menor do que o Java e o consumo inicial de processamento

Figura 32 – Fatorial (10) - Média

Fonte: Herik Lima de Castro

do *CWF* foi 20,33 vezes menor. Durante as requisições, o consumo de memória *RAM* do *CWF* foi 118,39 vezes menor do que o Java, o pico de consumo de processamento do *CWF* foi 5,22 vezes menor do que o Java e o *CWF* conseguiu responder às requisições 5,11 vezes mais rápido, como mostra a Figura 33.

Tabela 3 – Primo (997) - Resultados

	Teste 1		Teste 2		Teste 3	
	CWF	Java 8	CWF	Java 8	CWF	Java 8
RAM inicial (KB)	792,00	103.116,80	792,00	103.731,20	792,00	109.772,80
RAM final (KB)	1.433,60	141.004,80	1.126,40	158.105,60	1.126,40	150.528,00
Processador % (Ini)	0,00	0,00	0,00	0,00	0,00	0,00
Processador % (Fim)	14,00	56,00	17,00	75,00	17,00	59,00
Tempo Resposta (ms)	649,00	3.178,00	671,00	2.979,00	667,00	2.899,00

Fonte: Herik Lima de Castro

O terceiro teste consistia em verificar qual seria o número 20 na sequência de Fibonacci. A Figura 4 mostra os resultados obtidos em cada teste. Em média, o *CWF* teve um consumo inicial de memória *RAM* 132,22 vezes menor do que o Java e o consumo inicial

Figura 33 – Primo (997) - Média

Fonte: Herik Lima de Castro

de processamento do *CWF* foi 47,57 vezes menor. Durante as requisições, o consumo de memória *RAM* do *CWF* foi 104,61 vezes menor do que o Java, o pico de consumo de processamento do *CWF* foi 3.37 vezes menor do que o Java e o *CWF* conseguiu responder às requisições 4.57 vezes mais rápido, como mostra a Figura 34.

Tabela 4 – Fibonacci (20) - Resultados

	Teste 1		Teste 2		Teste 3	
	CWF	Java 8	CWF	Java 8	CWF	Java 8
RAM inicial (KB)	796,00	103.833,60	792,00	107.008,00	792,00	103.833,60
RAM final (KB)	1.433,60	137.420,80	1.228,80	138.342,40	1.228,80	131.276,80
Processador % (Ini)	0,00	52,00	0,00	52,00	0,00	39,00
Processador % (Fim)	19,00	70,00	20,00	62,00	20,00	67,00
Tempo Resposta (ms)	741,00	3.500,00	739,00	3.379,00	715,00	3.163,00

Fonte: Herik Lima de Castro

O quarto e último teste consistia em gerar uma página *HTML* dinâmica contendo em seu corpo os 10 mil caracteres. A Tabela 5 apresenta os resultados obtidos em cada teste. Em média, o *CWF* teve um consumo inicial de memória *RAM* 131,06 vezes menor do que o Java e o consumo inicial de processamento do *CWF* foi 25 vezes menor. Durante as requisições, o consumo de memória *RAM* do *CWF* foi 346,35 vezes menor do que o

Figura 34 – Fibonacci (20) - Média

Fonte: Herik Lima de Castro

Java, o pico de consumo de processamento do *CWF* foi 4,94 vezes menor do que o *Java* e o tempo de resposta de ambos foi muito similar, com uma ligeira vantagem para o *CWF*, como mostra a Figura 35.

Tabela 5 – Pages (10000) - Resultados

	Teste 1		Teste 2		Teste 3	
	CWF	Java 8	CWF	Java 8	CWF	Java 8
RAM inicial (KB)	792,00	104.345,60	792,00	102.502,40	792,00	104.550,40
RAM final (KB)	884,00	292.044,80	892,00	283.955,20	884,00	345.292,80
Processador % (Ini)	0,00	13,00	0,00	34,00	0,00	28,00
Processador % (Fim)	10,00	51,00	11,00	58,00	10,00	44,00
Tempo Resposta (ms)	9.157,00	9.400,00	9.199,00	9.482,00	9.223,00	9.503,00

Fonte: Herik Lima de Castro

De modo geral, o *CWF* consome menos memória, menos processamento e, em geral, possui um tempo de resposta melhor do que o *Java*. Em todos os testes, o consumo inicial de memória *RAM* foi cerca de 130 vezes menor no *CWF* e o consumo inicial de processamento chegou a ser até 52 vezes menor. Durante as requisições, o consumo de memória *RAM* utilizada pelo *CWF* chegou a ser mais de 390 vezes menor e o consumo de processamento chegou a ser mais de 5,5 vezes menor. Em alguns testes, o *CWF* conseguiu

Figura 35 – Pages (10000) - Média

Fonte: Herik Lima de Castro

atingir um tempo de resposta mais de 6 vezes menor do que o Java. Em testes com muita transferência de dados, o tempo de resposta tende a ficar parecido, uma vez que os servidores ficam limitados à velocidade da rede. Entretanto, com a evolução das tecnologias de rede, e com o aumento de sua velocidade, a diferença no tempo de resposta entre os servidores web poderá se tornar mais perceptível.

Outro fator importante a ser considerado nesta avaliação é o fato de que o *CWF* consome muito menos recursos computacionais do que o Java e, ainda assim, consegue ter tempos de respostas melhores ou semelhantes. Esse fator pode contribuir para a redução dos custos com servidores e também com o consumo de energia elétrica. Os processadores são conhecidos como os principais responsáveis pelo consumo de energia. Entretanto, estudos sugerem que o consumo de energia causado pelas memórias principais também podem variar entre 30% a 40% do consumo de energia gerado pelo servidor (BARROSO; HOELZLE, 2009). Estudos mais recentes apontam que o consumo de energia causado pelas memórias principais pode até mesmo superar o consumo de energia dos processadores (APPUSWAMY; OLMA; AILAMAKI, 2015). Neste contexto, o *CWF* pode ser uma alternativa para a

redução dos custos com servidores, mas avaliações específicas ainda precisam ser realizadas neste contexto.

Aqui será demonstrada uma avaliação de desempenho. Em atividades futuras, serão definidos os algoritmos e cenários mais apropriados para a comparação de desempenho e consumo de recursos computacionais entre *web frameworks*. Com o ambiente e os cenários definidos, o *CWF* será comparado novamente com o Java e com os *frameworks* de desenvolvimento de aplicações *web* em C++.

4.4 A avaliação de viabilidade e aplicabilidade

O *CWF* é um projeto de código-aberto hospedado no repositório do *GitHub* sob licenciamento *MIT*. Portanto, desenvolvedores podem usar o *web framework*, seja para submeter contribuições para corrigir defeitos ou inserir novas funcionalidades, seja para utilizá-lo no desenvolvimento de aplicações *web* em C++. Visto que o *CWF* atingiu um estágio em que pode ser usado para a construção de aplicações reais, uma avaliação foi realizada para entender se e como ele tem sido utilizado. Nesta seção são discutidas as avaliações realizadas com usuários reais do *CWF* e com as aplicações que já foram desenvolvidas.

4.4.1 Questionários aplicados a usuários do GitHub

Os usuários do *GitHub* podem baixar, modificar, atribuir uma *Star* (indicando que gostou do projeto) e/ou fazer um *Fork* (uma cópia do projeto) dos projetos. Por meio das *Stars* e dos *Forks*, é possível acessar o perfil dos usuários para contatá-los e obter algum *feedback* sobre o projeto. Desta forma, um questionário sobre o uso do *CWF* foi enviado para todos os desenvolvedores que interagiram com o repositório do *CWF*.

Este experimento foi planejado da seguinte forma:

- Sujeitos: qualquer usuário que tenha atribuído uma *Star* e/ou feito um *Fork* do *CWF* no repositório do *GitHub*.
- Contato com os sujeitos: acessar o perfil dos usuários que tenham atribuído uma *Star* e/ou feito um *Fork* do *CWF* no repositório do *GitHub* e tentar obter o *e-mail* de contato.

- Coleta de dados: a coleta dos dados será feita por meio de um formulário online, em inglês, feito no Google Formulários, com perguntas a respeito do tipo de aplicação desenvolvida, nível de dificuldade da instalação e uso do *CWF*, se usariam o *CWF* em um ambiente real e se estão dispostos a responder perguntas mais específicas a respeito do projeto que desenvolveram.
- Análise dos dados: utilização de estatísticas descritivas.

Durante a execução deste estudo, foram enviados *e-mails* para os usuários cujos *e-mail* de contato estavam disponíveis em seu perfis. Ao todo, foram acessados os perfis de 270 usuários do GitHub que atribuíram uma *Star* e/ou fizeram um *Fork* do *CWF*. Dos 270 perfis acessados, foram obtidos os e-mails de 110 usuários, dos quais apenas 3 responderam ao questionário.

A primeira pergunta se refere ao tipo de aplicação que foi desenvolvida. Dois dos usuários desenvolveram aplicações mais complexas para ver como o *CWF* se sairia e um desenvolveu uma aplicação mais simples.

A segunda pergunta foi referente ao tipo de aplicação que desenvolveram e o propósito geral da aplicação. O primeiro desenvolveu uma aplicação de demonstração. O segundo criou um *web site* para negócios pessoais, e o terceiro criou um *blog* pessoal, usado principalmente para compartilhar dificuldades técnicas.

A terceira pergunta se refere ao nível de dificuldade para se instalar o *C++ Web Framework*. Dois participantes consideraram o nível de dificuldade como **Fácil** e o outro como **Muito Fácil**.

A quarta pergunta foi opcional e se refere às dificuldades de instalação do *C++ Web Framework*, mas não foi reportado nenhum problema relacionado à instalação do *web framework*.

A quinta pergunta foi opcional a respeito de comentários a respeito do processo de criação de aplicações utilizando o *C++ Web Framework*. Apenas um usuário respondeu e disse que é simples se criar aplicações utilizando o *C++ Web Framework*.

A sexta pergunta foi opcional e era a respeito de comentários sobre a manutenibilidade das aplicações utilizando o *C++ Web Framework*, mas nenhum dos 3 usuários respondeu.

A sétima pergunta foi opcional e se referia ao desempenho do *C++ Web Framework*. Dois usuários responderam; um deles não realizou nenhum teste específico de desempenho e o outro achou muito rápido.

A oitava pergunta foi opcional e pedia para que os usuários descrevessem os pontos fortes do *C++ Web Framework*. O primeiro disse que é fácil e rápido. O segundo disse que o *CWF* se beneficia de todo o ferramental no mundo do *Qt* e do C++, e é muito melhor capturar a maioria dos erros potenciais com compiladores, analisadores estáticos, sanitizantes de endereço, etc, em vez de executar um teste de unidade para verificar que uma *string* é uma *string*. O último disse que a maior vantagem do *CWF* é, provavelmente, o fato de ter sido escolhido o *Qt Framework* como *framework* de apoio.

A nona pergunta foi opcional e pedia para que os usuários descrevessem os pontos fracos do *C++ Web Framework*. O primeiro deles reportou problemas com a documentação e ausência de uma comunidade para se discutir e aprender mais sobre o *web framework*. O segundo acredita que devemos otimizar o site oficial do *CWF* e pediu para que fossem expostos mais exemplos de aplicações utilizando o *CWF* para ajudar no aprendizado. O último pediu uma ferramenta *ORM* integrada ao *CWF* e suporte à *FastCGI*.

A décima pergunta pediu para que os usuários respondessem se utilizariam o *C++ Web Framework* no desenvolvimento de aplicações reais. Um deles disse que sim, enquanto dois talvez utilizariam.

A décima primeira pergunta pediu para que os usuários justificassem as respostas da pergunta anterior. O primeiro disse que atualmente ele é um Engenheiro de Softwares Java, mas que, no passado, ele aprendeu C++ e tem um grande interesse em desenvolver aplicações *web* utilizando C++. O segundo disse que, atualmente, já está desenvolvendo uma aplicação real utilizando o *CWF* e o último não entendeu a pergunta.

Por fim, perguntamos se eles estariam dispostos a responder mais questões a respeito do *C++ Web Framework* ou participar de um estudo acadêmico conduzido online e todos disseram que sim.

Embora poucos desenvolvedores tenham respondido ao questionário, as respostas obtidas foram bastante positivas e algumas das críticas serão levadas em consideração, principalmente no que se refere à documentação fraca, ausência de um fórum e disponibilização de mais exemplos.

4.4.2 Uso em ambiente de produção real

Nesta seção, dois usos particulares do *CWF* para o desenvolvimento de aplicações reais são descritos. A primeira é uma aplicação real criada pelo próprio autor do *web framework* e a segunda é uma aplicação criada por um colaborador que encontrou o *CWF* no *GitHub* e, desde então, tem usado para desenvolver suas aplicações.

Web service de Contas a Pagar

O autor deste projeto de pesquisa trabalha na AGIT Informática LTDA, uma empresa que fornece consultoria, treinamento e desenvolvimento de softwares para empresas de diversos setores. Um dos clientes é um sindicato, portanto, um *ERP* feito em C++ com *Qt* e *PostgreSQL* foi projetado especificamente para atender esse tipo de empresa, que possuem regras bem particulares. Um dos maiores módulos do *ERP* é o Contas a Pagar, onde os funcionários lançam notas fiscais de produtos e serviços contratados pelo sindicato para que o sistema faça retenção de impostos, pagamento via FEBRABAN e envie os dados para a contabilidade a respeito desses lançamentos.

Recentemente, este cliente contratou uma empresa para desenvolver um Contas a Pagar simplificado para as suas 110 subsedes. Este sistema possui menos informações do que o Contas a Pagar que é utilizado na sede central. Isso se deve ao fato de que o Contas a Pagar utilizado na sede central é muito complexo, principalmente no que se refere à retenção de impostos, pois existem inúmeros tipos de serviços, várias categorias de empresas e tudo isso influencia na retenção dos impostos.

Por falta de pessoal com conhecimento em regras contábeis em todas as suas subsedes, eles optaram por desenvolver um Contas a Pagar simplificado, que ainda está em fase de testes em apenas duas subsedes. Nesse Contas a Pagar simplificado, os usuários precisam apenas manter um cadastro de fornecedores e fazerem lançamentos das notas fiscais, informando o fornecedor, valor, data de emissão, data de vencimento, data de pagamento, desconto, juros e multa. Todas essas informações precisam ser repassadas à sede central de modo que ela possa gerenciar todos os gastos feitos pelas suas subsedes.

Como os sistemas foram desenvolvidos por duas empresas diferentes, e precisavam se comunicar, optou-se por desenvolver um *web service REST* em *CWF*, onde o Contas a

Pagar simplificado envia todos os seus lançamentos feitos para o Contas a Pagar hospedado na sede central. O servidor com *web service REST* foi instalado em 19 de dezembro de 2018 em um dos servidores do CPD da sede central, e está em operação até hoje. Entretanto, como o Contas a Pagar simplificado ainda está em fase de testes, apenas alguns testes foram realizados para verificar a viabilidade do *CWF*. Nos testes, o Contas a Pagar simplificado já exportou cerca de 2 mil lançamentos para o Contas a Pagar da sede central, utilizando o *web service REST*, feito utilizando o *CWF*. As informações são enviadas utilizando o formato *JSON* e são validadas pelo *web service* escrito no *CWF*.

A Figura 36 mostra o funcionamento da aplicação desenvolvida. No centro da imagem, temos o servidor que está na sede central, com o *web service* em *CWF* sendo executado e recebendo requisições de computadores das subsedes por meio do Contas a Pagar simplificado.

Figura 36 – Funcionamento da aplicação

Fonte: Herik Lima de Castro

Caso tudo esteja certo, os dados são inseridos no banco de dados e uma mensagem em formato *JSON* é enviada como resposta. Se houver alguma inconsistência nas informações enviadas, o *web service* também envia uma resposta utilizando *JSON*, que informa onde o erro ocorreu. Até o momento, o *web service* mostrou-se bastante rápido e estável nos testes realizados. Futuramente, haverá mais informações a respeito do desempenho dele quando o Contas a Pagar simplificado for instalado em todas as subsedes. Quando isso acontecer, é esperado um volume de, no mínimo, mil lançamentos por dia sendo enviados ao *web service*. Embora ele não tenha operado em ambiente de

intenso uso, a robustez e a estabilidade fornecida pela versão atual do *web framework* deu confiança à empresa em adotar o *CWF* para o desenvolvimento deste *web service*. Devido ao bom funcionamento do *web service* desenvolvido em *CWF*, um novo *web service* já foi desenvolvido para integrar dados de Receita (valores a receber) ao Contas a Receber, dados esses que virão do sistema de Caixa, Nethotel, Convênios e Mensalidades. O Caixa e Nethotel já estão prontos e entram em funcionamento em setembro de 2019.

SymAlgo Technologies

O presidente da empresa francesa *SymAlgo Technologies*¹ desenvolveu aplicações reais utilizando o *CWF* e relatou suas impressões sobre o *framework*. A empresa fornece serviços de desenvolvimento de software, modernização de aplicativos e prototipação de ideias. Ele utilizou o *CWF* em 4 projetos reais, sendo dois sites de apresentação que podem ser acessados pelos *links*: $\langle \text{https://www.symalgo-tech.com/en} \rangle$ e $\langle \text{https://naratym.com/fr} \rangle$. O terceiro projeto é tanto um site, quanto uma aplicação web *standalone*, que não está 100% completo, mas já oferece uma série de funcionalidades e pode ser acessado pelo *link* $\langle \text{https://app.actograph.io/\#/login} \rangle$. A quarta aplicação trata-se de um protótipo de uma aplicação web para testar redes neurais, onde o *CWF* é utilizado como back-end para controlar a rede neural. Algumas das aplicações desenvolvidas por ele, usando o *CWF*, são novas versões de aplicações que ele já havia desenvolvido em sua organização e achou melhor migrar para uma plataforma em C++. Outras são soluções desenvolvidas para uso interno e para terceiros.

Em sua opinião, o *CWF* é fácil de configurar e de instalar, mas não atende por completo todas as necessidades que ele precisa, pois, em seu trabalho, ele precisa desenvolver aplicações *standalone* e aplicações web. Ele fez uma comparação do *CWF* com o *NodeJS* e ele acredita que seria interessante que o *CWF* implementasse nativamente alguns módulos de alto-nível, como o *NodeJS*, como, por exemplo, módulo `graphql`¹. No entanto, ele escolheu o *CWF* porque pôde facilmente adapta-se às suas necessidades. Ele também considerou o desenvolvimento de aplicações com o *CWF*, um pouco mais complexo do que com o *NodeJS*, pela ausência de alguns módulos que o *NodeJS* disponibiliza e o *CWF* ainda não.

¹ <https://www.symalgo-tech.com/en>

Com relação à facilidade de manter uma aplicação com o *CWF*, ele considera mais fácil embutir todo o código do *CWF* na mesma aplicação ao invés de criar uma biblioteca dinâmica. Com relação ao tempo de respostas, consumo de memória *RAM* e processamento, o autor considera similar ao *NodeJS*, mas muito superior ao *Symfony*. Como pontos fortes do *CWF*, o autor mencionou os seguintes pontos: *design* (simples e limpo), C++ (Estrutura, velocidade, adaptabilidade) e o uso do *Qt* (muitas ferramentas para serem usadas). Como pontos fracos, o autor mencionou que o *CWF* precisa de mais módulos e funcionalidades, como o *graphql*, por exemplo. Ao final, o autor disse que o projeto tem um grande potencial de crescimento e que ele também é um bom argumento para se dizer que o C++ pode ser uma boa escolha para se desenvolver aplicações web modernas, mas que ele tem a sensação de que a ausência de mais módulos pode dificultar o uso para os novos usuários.

4.5 Página web e divulgação

No final de 2016, o *CWF* foi inserido em um repositório público no *GitHub*². Embora o *GitHub* só armazene dados de acesso dos últimos 15 dias, é possível notar que o número de acesso tem sido alto nos últimos meses com a evolução do *web framework*. Por exemplo, entre os dias 15 de agosto de 2019 a 28 de agosto de 2019 houve 617 visualizações de 98 visitantes únicos, como mostra a Figura 37. O repositório do *CWF* no *GitHub* também ganhou 290 estrelas de pessoas de 30 diferentes países e 51 usuários fizeram uma cópia (*fork*) do repositório para altera-lá, sem modificar o repositório original.

² <https://github.com/HerikLyma/CppWebFramework>

Figura 37 – Número de acessos no *GitHub*

Fonte: Herik Lima de Castro

Além disso, uma página web foi criada para o *CWF*³ em janeiro de 2018 e já conta com 2 mil acessos.

4.6 Considerações finais

A primeira avaliação não controlada, em que foi pedido para que os participantes instalassem o *web framework* e desenvolvessem algum tipo de aplicação com ele, foi o nosso primeiro indício de que o protótipo era fácil de instalar e configurar. Depois disso, foram elaborados experimentos controlados com usuários, que, embora possuam limitações de generalização por fatores como quantidade de participantes, didática do instrutor e o nível da turma, os resultados novamente deram indícios de que o protótipo criado, é, de fato, fácil de se instalar e utilizar. Os fatores relacionados à facilidade de uso também foram reforçados pelas respostas dos usuários do *GitHub*, e as aplicações reais desenvolvidas utilizando o protótipo mostram a sua viabilidade. Embora estejam previstos novos testes para o futuro, esse conjunto de testes já realizados indicam que o protótipo proposto tem potencial e cumpre seus objetivos de mitigar ou resolver problemas relacionados ao desenvolvimento de aplicações *web*. Os testes de desempenho possuem limitações em razão de alguns fatores, mas entende-se que é necessário ter uma ideia do desempenho do protótipo, e para essa comparação, foi selecionada a plataforma que mais fornecia funcionalidades similares ao *CWF*, e era o mais usado em sites de alto-desempenho o Java EE.

³ www.cppwebframework.com

5 Conclusões

O desenvolvimento de aplicações *web* pode ser intimidador até mesmo para programadores experientes em razão da complexidade dos *frameworks* disponíveis ou da difícil manutenção das aplicações desenvolvidas com alternativas mais simples que, geralmente, misturam código da camada de negócios com código da camada de aplicação. Além disso, a arquitetura das aplicações *web* e os recursos oferecidos pelos *frameworks* disponíveis não favorecem as aplicações que requerem maior desempenho e uso econômico de recursos computacionais.

Neste contexto, o objetivo desta dissertação foi propor o desenvolvimento e avaliação de um *web framework* para o desenvolvimento de aplicações *web* em C++ que seja simples de usar, fácil de configurar, apresente bom desempenho e seja econômico quanto ao uso de recursos computacionais.

Nas avaliações de uso do *web framework*, há indícios de que ele seja fácil de usar, visto que os participantes dos estudos realizados conseguiram executar as tarefas propostas sem dificuldades. Esses resultados foram bastante satisfatórios, principalmente se levar em conta que o perfil dos participantes, em que 50% deles nunca havia tido contato com o desenvolvimento *web* e 20% deles nunca havia programado em C++. O último teste realizado com usuários foi dividido em duas partes, sendo a primeira igual à anterior e uma atividade extra relacionada à *web services REST*, para que os participantes resolvessem em casa. A primeira etapa deste experimento foi satisfatória e refletiu dados muito similares às avaliações anteriores. Já a atividade com *web services REST*, que os participantes deveriam realizar em casa, se mostrou insatisfatória, uma vez que menos da metade dos participantes conseguiu desenvolver. No entanto, existem indícios de falta de empenho dos usuários com relação às atividades designadas para resolver em casa, uma vez que foi necessário insistir para que os participantes finalizassem o experimento e submetessem os dados coletados.

Em relação ao desempenho e uso de recursos computacionais, o *C++ Web Framework (CWF)* teve um desempenho superior ao Java nos quesitos de tempo de resposta, consumo de memória e processamento, chegando a consumir até 52 vezes menos processamento durante a inicialização e até 5,53 vezes menos processamento durante as requisições. Com relação ao consumo de memória *RAM*, o *CWF* consome tipicamente 130 vezes menos

memória do que o Java, mas esse número chegou a até 390,6 vezes menos durante as requisições. No tempo de resposta, o *CWF* chegou a ser até 6,03 vezes mais rápido do que o Java, mas no teste “Pages”, um teste que envolvia a construção de uma página web dinâmica com uma grande quantidade de bytes, foi constatado que o *CWF* consumiu em média 346,35 vezes menos memória RAM e 4,94 vezes menos processamento. Embora o *CWF* tenha tido um tempo de resposta melhor do que o Java em todos os testes, foi percebido uma queda na diferença do tempo de resposta entre ambos quando se aumenta a quantidade de dados a serem trafegados. Isso se deve às limitações de velocidade de rede, mas mesmo nesses casos, o *CWF* consumiu menos memória e processamento para realizar a mesma atividade que o Java.

O uso do *CWF* em aplicações reais mostram a viabilidade e o potencial do *framework*. Destaca-se o uso pela empresa francesa SymAlgo Technologies na criação de soluções internas e para terceiros. Em geral, todos os que utilizaram o *CWF* para a criação de aplicações reais o acharam fácil de instalar, configurar e criar aplicações.

Embora a avaliação de uso, desempenho e uso de recursos computacionais tenham sido satisfatórios, acredita-se que experimentos mais elaborados precisam ser realizados, principalmente, em comparação a outras soluções. Entretanto, os resultados atuais foram úteis para motivar a continuidade do projeto e construir um *web framework* robusto, simples de se usar e configurar, e que é econômico no consumo de memória *RAM* e processamento, além de apresentar um desempenho satisfatório no tempo de resposta. O *CWF* pode ser uma boa alternativa para a criação de aplicações *web* utilizando C++, principalmente para aqueles que não possuem ou não têm disponibilidade para adquirir conhecimento em outras linguagens de programação.

5.1 Resultados

O principal resultado deste projeto de pesquisa foi a construção e a avaliação de um *framework web* para o desenvolvimento de aplicações *web* em C++ chamado *CWF* (*C++ Web Framework*). Este *framework* pode ser usado tanto para a criação de aplicações *web* tradicionais quanto para a construção de serviços *web*. O *web framework* é multiplataforma, possui um servidor *web* nativo e induz o usuário a utilizar a arquitetura *MVC* (*Model-View-Controller*). Ele também fornece suporte à manipulação de *JSON* e *XML*, acesso a

banco de dados relacionais e à criação de *web services REST*. O *CWF* foi construído com base no Qt, que é um *framework* C++ multiplataforma e que já fornece suporte a muitas áreas comuns ao desenvolvimento de softwares, como rede, *XML*, *JSON*, *SQL*, manipulação de arquivos e etc. Sendo assim, o novo *web framework* aproveita uma série de recursos que vêm sendo utilizados há mais de 20 anos por mais de 1 milhão de desenvolvedores ao redor do mundo.

5.2 Contribuições

Este projeto pretende contribuir com a área de desenvolvimento de software de diversas formas. Em primeiro lugar, estudando formas de desenvolver *web frameworks* mais usáveis, rápidos, eficientes e que funcionem em uma grande variedade de dispositivos e sistemas operacionais diferentes, aplicando técnicas estudadas no referencial bibliográfico. É esperado que o novo *web framework* não imponha barreiras aos desenvolvedores inexperientes e, que, na visão dos usuários, por meio de pesquisas, ele seja considerado simples de usar e fácil de configurar.

Em princípio, acredita-se que as aplicações web mais simples sejam as mais adequadas para serem implementadas com este *web framework* porque nenhum estudo de larga escala será realizado para avaliá-lo. Em segundo lugar, o *web framework* proposto pode ser útil para uma ampla população de desenvolvedores C++ que desejam criar aplicações web sem precisar aprender outras linguagens de programação. Embora existam outras alternativas para desenvolvimento web em C++, espera-se que a simplicidade e o uso de boas práticas que permitem uma boa manutenção e evolução sejam atraentes aos desenvolvedores. É esperado que o novo *web framework* forneça suporte para a criação de aplicações ou funcionalidades específicas que necessitam de um bom desempenho e que forneça meios de se criar aplicações web tradicionais e também serviços web. Por fim, este trabalho também traz consigo reflexões e decisões de projeto sobre o desenvolvimento de *web frameworks* que poderão ser utilizadas por todos os desenvolvedores de modo a guiar melhor a área e contribuir para que novas técnicas e novos *web frameworks* sejam desenvolvidos para aumentar a produtividade dos desenvolvedores, reduzir a exigência por servidores mais potentes para se executar aplicações web e tornar o desenvolvimento web mais simples.

5.3 *Trabalhos futuros*

Algumas atividades foram previstas como trabalhos futuros. Em primeiro lugar, pretende-se realizar mais avaliações sobre a instalação, configuração e uso do *CWF* em comparação com outras alternativas, como o Java EE. Ao invés de avaliar apenas se os usuários conseguiram realizar as tarefas e em quanto tempo, pretende-se comparar os resultados com grupos de controle executando as mesmas atividades utilizando outro *web framework*. Pretende-se, também, realizar novos testes de desempenho utilizando aplicações com mais processamento de grandes volumes de dados. Por fim, pretende-se definir métricas para comparar a manutenibilidade das aplicações criadas em *CWF* em comparação com outras soluções existentes.

Referências¹

- ADOBE. *ColdFusion*. 2016. Disponível em: <https://www.adobe.com/br/products/coldfusion-enterprise.html>. Citado na página 25.
- APPUSWAMY, R.; OLMA, M.; AILAMAKI, A. Scaling the memory power wall with dram-aware data management. *ACM*, In Proceedings of the 11th International Workshop on Data Management on New Hardware (DaMoN'15). ACM, New York, NY, USA, 3:1–3:9., 2015. Disponível em: <https://dl.acm.org/citation.cfm?doid=2771937.2771947>. Citado na página 85.
- BARROSO, L.; HOELZLE, U. The datacenter as a computer: An introduction to the design of warehousescale. *Morgan Claypool*, 2009. Citado na página 85.
- BASHAM, B.; SIERRA, K.; BATES, B. *Head First Servlets and JSP*. 2. ed. [S.l.]: O'Reilly, 2008. Citado na página 27.
- BJÖREMO, M.; TRNINIĆ, P. *Evaluation of web application frameworks*. Dissertação (Mestrado) — University of Gothenburg, 2010. Citado 2 vezes nas páginas 26 e 33.
- BOWEN, R.; COAR, K. *Apache Cookbook*. 3. ed. [S.l.]: O'Really, 2018. Citado na página 22.
- BROWN, D.; DAVIS, C. M.; STANLICK, S. *Struts 2 in Action*. 1. ed. [S.l.]: Manning Publications, 2008. Citado na página 26.
- BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, v. 25, p. 599–616, 2009. Citado na página 17.
- CHARLAND, A.; LEROUX., B. Mobile application development: Web vs. native. *Future Generation Computer Systems*, v. 9, 2011. Citado na página 17.
- CHAUBEY, R.; SURESH, J. K. Integration vs. development: an engineering approach to building web applications. *IEEE Multimedia*, IEEE Multimedia, p. 171–181, 2001. Citado na página 17.
- CHOCHLIK, M. Portable reflection for c++ with mirror. *Journal of Information and Organizational Sciences*, 2012. Citado na página 34.
- COMPANY, Q. *Qt*. 2019. Disponível em: <https://www.qt.io/>. Citado na página 39.
- DECK, P. *Spring MVC: A Tutorial (English Edition)*. [S.l.]: Brainy Software, 2008. Citado na página 26.
- DUMON, W.; DEFORCHE, K. Wt: A web toolkit. *Dr. Dobb's Journal*, v. 33, p. 55–59, 2008. Citado na página 28.
- EZUST, A.; EZUST, P. *An Introduction To Design Patterns In C++ With Qt 4*. 1. ed. [S.l.]: Prentice Hall, 2006. Citado na página 34.

¹ De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

- FJORDVALD, M.; NEDELCU, C. *Nginx HTTP Server - Fourth Edition: Harness the power of Nginx to make the most of your infrastructure and serve pages faster than ever before*. [S.l.]: Packt Publishing - ebooks Account, 2018. Citado na página 22.
- FOURMENT, M.; GILLINGS, M. R. A comparison of common programming languages used in bioinformatics. *BMC Bioinformatics*, Springer, 2008. Citado na página 33.
- FRINGS, S. *QtWebApp HTTP Webserver in C++*. 2010. Disponível em: <http://stefanfrings.de/qtwebapp/index-en.html>. Citado na página 30.
- GAME, T. C. L. B. 2018. Disponível em: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/>. Citado 2 vezes nas páginas 19 e 36.
- GEARY, D.; HORSTMANN, C. *Core JavaServer Faces*. 3. ed. [S.l.]: Prentice Hall, 2010. Citado na página 27.
- GINIGE, A.; MURUGESAN, S. Web engineering: An introduction. *IEEE Multimedia*, IEEE Multimedia, v. 8, p. 14–18, 2001. Citado na página 17.
- GONCALVES, A. *Beginning Java EE 7*. 1. ed. [S.l.]: Apress, 2013. Citado na página 27.
- GOOGLE. *Google Trends*. 2016. Disponível em: <https://trends.google.com.br/trends/explore?date=2016-01-01%202016-12-31&q=%2Fm%2F034w42,%2Fm%2F069fr,STXXL,%2Fm%2F091284>. Citado na página 38.
- HOREK, K. *Learning Zurb Foundation*. [S.l.]: Packt Publishing, 2014. Citado na página 26.
- HUNDT, R. *Loop Recognition in C++/Java/Go/Scala*. 1600 Amphitheatre Parkway Mountain View, CA, 94043, 2011. Citado 2 vezes nas páginas 32 e 36.
- HUNTER, J.; CRAWFORD, W. *Java Servlet Programming*. 2. ed. [S.l.]: O'Reilly, 2001. Citado na página 27.
- JETBRAINS. *C++ annotated: Summer edition*. 2015. Disponível em: <https://blog.jetbrains.com/clion/2015/09/cpp-annotated-summer-edition/>. Citado na página 37.
- KHAN, A. *Microsoft IIS 10.0 Cookbook*. [S.l.]: Packt Publishing, 2017. Citado na página 22.
- LUBIN, M.; DUNNING, I. Computing in operations research using julia. *INFORMS*, MIT Operations Research Center, 2015. Citado 2 vezes nas páginas 18 e 34.
- MILLARES, G. *Top 5 Programming Languages Used In Web Development*. 2015. Disponível em: <http://blog.stoneriverelearning.com/top-5-programming-languages-used-in-web-development/>. Citado 3 vezes nas páginas 19, 79 e 80.
- NEVES, P.; PAIVA, N.; DURÃES, J. A comparison between java and php. *Proceeding C3S2E '13 Proceedings of the International Conference on Computer Science and Software Engineering*, p. 130–131, 2013. Citado na página 33.
- OBILTSCHNIG, G. *POCO C++ Libraries*. 2005. Disponível em: <https://pocoproject.org/>. Citado na página 31.

- OFFUTT, J. Quality attributes of web software applications. *IEEE Multimedia*, IEEE Multimedia, v. 19, p. 25–32, 2002. Citado na página 17.
- PAUTASSO, C. Restful web services: Principles, patterns, emerging technologies. *Springer*, Springer, p. 31–51, 2013. Citado na página 22.
- PRABHU, A.; SHENOY, A. *Introducing Materialize*. 1. ed. [S.l.]: O’Reilly, 2016. Citado na página 26.
- PRECHELT, L. An empirical comparison of C, C++, Java, P *+—rl, Python, Rexx, and Tcl. *IEEE Computer*, IEEE Computer, 2000. Citado na página 33.
- PRECHELT, L. Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Elsevier*, Elsevier, v. 57, p. 205–270, 2003. Citado na página 33.
- RAEMAEEKERS, S.; DEURSEN, A. van; VISSER, J. Semantic versioning versus breaking changes: A study of the maven repository. *IEEE International Working Conference on Source Code Analysis and Manipulation*, 2014. Citado na página 18.
- RAMANA, U. V.; PRABHAKAR, T. V. Some experiments with the performance of lamp architecture. *IEEE*, IEEE, p. 916–920, 2005. Citado 2 vezes nas páginas 32 e 36.
- SALAS-ZARATE, M. del P. et al. Analyzing best practices on web development frameworks: The lift approach. *Science of Computer Programming*, v. 102, p. 1–19, 2015. ISSN 0167-6423. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167642314005735>. Citado 3 vezes nas páginas 17, 18 e 40.
- SALNIKOV-TARNOVSKI, N. *Most Popular Java Application Servers (2017 Edition)*. 2017. Disponível em: <https://dzone.com/articles/most-popular-java-application-servers-2017-edition>. Citado na página 80.
- SCHWARZ, M. *Design and Analysis of Web Application Frameworks*. Tese (Doutorado) — Aarhus University, 2013. Citado na página 26.
- SHKLAR, L.; ROSEN, R. *Web Application Architecture Principles, protocols and practices*. [S.l.]: Wiley, 2004. Citado 6 vezes nas páginas 18, 22, 23, 24, 25 e 40.
- SINHA, N.; KARIM, R.; GUPTA, M. Simplifying web programming. In: *Proceedings of the 8th India Software Engineering Conference, ISEC 2015, Bangalore, India, February 18-20, 2015*. [S.l.: s.n.], 2015. p. 80–89. Citado 5 vezes nas páginas 17, 18, 28, 34 e 40.
- SPIELMAN, S. *JSTL Practical Guide for JSP Programmers*. [S.l.]: The Morgan Kaufmann Practical Guides Series, 2003. Citado na página 27.
- SPURLOCK, J. *Bootstrap: Responsive Web Development*. 1. ed. [S.l.]: O’Reilly, 2013. Citado na página 26.
- SRAI, A. et al. Applying mda approach for spring mvc framework. *International Journal of Applied Engineering Research*, MIT Operations Research Center, v. 12, p. 4372–4381, 2017. Citado na página 19.
- SWAIN, N. R. et al. A new open source platform for lowering the barrier for environmental web app development. *Environmental Modelling & Software*, v. 85, n. 1, p. 11–26, 2016. ISSN 1364-8152. Citado 3 vezes nas páginas 18, 28 e 34.

TECHEMPOWER. *TechEmpower Web Framework Benchmarks*. 2018. Disponível em: <https://www.techempower.com/benchmarks>. Citado na página 80.

TIOBE. *Index for August 2019*. 2019. Disponível em: <https://www.tiobe.com/tiobe-index/>. Citado 2 vezes nas páginas 19 e 37.

TRENT, S. et al. Performance comparison of php and jsp as server-side scripting languages. *Springer*, Springer, v. 5346, p. 164–182, 2002. Citado na página 32.

TSAI, W.; BAI, X.; HUANG, Y. Software-as-a-service (saas): Perspectives and challenges. *Science China Information Sciences*, v. 57, n. 5, p. 1–15, 2014. Citado na página 17.

VUORIMAA, P. et al. Leveraging declarative languages in web application development. *Springer*, v. 19, 2016. Citado 3 vezes nas páginas 17, 18 e 40.

WATTS, J.; GONZÁLEZ, J. *CakePHP 2 Application Cookbook*. [S.l.]: Packt Publishing, 2014. Citado na página 26.

ZHANG, T. et al. Tuning the aggressive tcp behavior for highly concurrent http connections in intra-datacenter. *IEEE/ACM Transactions on Networking*, IEEE, v. 25, p. 3808–3822, 2017. Citado na página 22.